Pythonov modul os i rad s procesima: izrada priručnika s primjerima

Roža, Marko

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Marko Roža

Pythonov modul os i rad s procesima: izrada priručnika s primjerima

Završni rad

Mentor: Doc. dr. sc. Vanja Slavuj

Rijeka, rujan 2021.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 1. veljače 2021. godine

Zadatak za završni rad

Pristupnik: Marko Roža

Naziv završnog rada:

Pythonov modul os i rad s procesima: Izrada radnog priručnika s primjerima

Naziv završnog rada na eng. jeziku:

Python os module and process management: A manual with examples

Sadržaj zadatka:

Proučiti, opisati i na vlastitim primjerima prikazati funkcionalnosti koje su povezane s operacijskim sustavom i ponuđene u okviru modula os programskoga jezika Python. Naglasak je potrebno staviti na funkcije i podatke koji se odnose na rad s procesima u operacijskome sustavu.

Mentor

Doc. dr. sc. Vanja Slavuj

Voditelj za završne radove

Dr. sc. Miran Pobar

Zadatak preuzet: 1. veljače 2021.

(potpis pristupnika)

Sažetak

U ovom diplomskom radu opisano je korištenje funkcija iz modula os programskog jezika Python. Kako bi primjena modula os bila što jasnije prikazana, objašnjeni su i temeljni pojmovi operacijskog sustava. U radu, funkcije modula os grupirane su prema njihovoj namjeni te su prikazani i detaljno objašnjeni primjeri upotrebe.

Ključne riječi: Linux**,** modul os, operacijski sustav, procesi, Python.

Sadržaj

1. Uvod

Operacijski sustav je softverski sloj kojim se nadograđuje računalni sustav. On povezuje i upravlja ostalim dijelovima računalnog sustava te olakšava i pojednostavljuje programerima, korisnicima i aplikativnim programima rad s računalnim sustavom (Tanenbaum, 2015). Funkcionalnosti operacijskog sustava proširuju se i unapređuju programiranjem sistemskih programa. Sistemski programi omogućuju operacijskom sustavu poboljšano upravljanje memorijom, procesima, ulaznim i izlaznim tokovima te pristup pohranjenim podacima organiziranim pomoću direktorija i datoteka (Techopedia, 2021). Osim sistemskih programa, u računalnom se sustavu izvode i aplikativni programi, koji su namijenjeni za rad korisnika. Aplikativni i sistemski programi, osim po namjeni, razlikuju se i po načinu izvođenja. Sistemski programi izvode se u kernel načinu rada, u kojem imaju pristup svim resursima računala, dok aplikativni programi rade u korisničkom načinu rada kako bi im se ograničio pristup određenim dijelovima računalnog sustava (Kovačić, 2008). Za upravljanje funkcijama operacijskog sustava koriste se razni programski jezici, poput programskih jezika C (u kojem su operacijski sustavi često i programirani), Python i raznih drugih.

Cilj ovog rada je prikazati i opisati mogućnosti upravljanja raznim funkcijama operacijskog sustava pomoću programskog jezika Python i njegovog modula os. U ovom radu objašnjeni su i određeni koncepti na kojima se temelji rad operacijskog sustava. U drugom poglavlju napravljen je uvod u programski jezik Python i njegov modul os. U trećem poglavlju objašnjen je koncept rada procesa i mogućnosti upravljanja njime pomoću modula os. Četvrto poglavlje govori o načinu na koji operacijski sustav pohranjuje podatke, a u petom poglavlju je riječ o ulaznim i izlaznim tokovima podataka i kako procesi mogu upravljati njima. Na kraju je dan zaključak rada.

2. Programski jezik Python i modul os

Python je objektno orijentirani, interpretirani programski jezik opće namjene kojeg je stvorio Guido von Rossum i prvi put je izdan 1991. godine. Stekao je popularnost zbog jednostavne sintakse, široke standardne biblioteke, podržavanja modula i paketa, skalabilnosti, otvorenog koda i stabilnosti (Python Software Foundation, What is Python? Executive Summary, 2021). Prema TIOBE indeksu iz svibnja 2021. godine (TIOBE Software BV, 2021), Python je drugi najpopularniji programski jezik i koristi se na raznim područjima, od poslovne informatike i obrazovanja do inženjerstva i prirodnih znanosti (Python Software Foundation, Python Success Stories, 2021). Jedna od najvažnijih karakteristika Pythona je široka standardna biblioteka koja se sastoji od više od 200 modula. Moduli su Python objekti kojima se klase, funkcije i varijable odvajaju u posebne cjeline (prema namjeni) i posebne su .py datoteke koje se moraju učitati kako bi se koristile. Korisni su za lakše razumijevanje i brže pisanje programskog koda. Python nudi mogućnost učitavanja cijelog modula ili samo određenih objekata, te stvaranje vlastitih modula (Python Software Foundation, What is Python? Executive Summary, 2021). Bitno je spomenuti i PEP (Python Enhancement Proposal) dokumente, kojima se objašnjavaju nove značajke uvedene u Python. Jedan od poznatijih PEP dokumenta je PEP-8 (Python Software Foundation, PEP 8 -- Style Guide for Python Code, 2021) kojim su utvrđena pravila pisanja programskog koda u programskom jeziku Python. Primjeri prikazani u sljedećim poglavljima nastoje pratiti PEP-8 pravila pisanja. U ovom trenutku, Python 3.9.6 je najnovija inačica ovog programskog jezika.

Slika 1. Logo programskog jezika Python

(preuzeto s https://www.python.org/static/community_logos/python-logo-master-v3-TM.png)

Modul os sadržan je u standardnoj biblioteci Pythona i pruža funkcionalnosti vezane za operacijski sustav. Njegova glavna namjena je olakšavanje interakcije korisnika i korisničkih programa s operacijskim sustavom što se može postići na različite načine, primjerice stvaranjem procesa i njihovim upravljanjem, prikupljanjem informacija o sustavu te upravljanjem datotečnim sustavom (Python Software Foundation, os — Miscellaneous operating system interfaces, 2021). Može se kazati da je modul os portabilan jer je većina njegovih funkcija dostupna neovisno o operacijskom sustavu ali određene funkcije (većinom vezane za upravljanje procesima) dostupne su jedino na operacijskim sustavima sličnim UNIX-u (Miletić, 2021).

U svim primjerima koji su opisani u ovome radu korištena je inačica 33 distribucije Fedora operacijskog sustava Linux. Verzija kernela operacijskog sustava je 5.11.10.

3. Upravljanje procesima

Proces je jedan od temeljnih koncepta operacijskog sustava. To je instanca programa koji se izvodi i sadrži sve podatke potrebne za njegovo izvođenje. Moderni operacijski sustavi za interaktivne sustave (u koje spadaju i osobna računala) podržavaju multiprogramiranje, istovremeno izvođenje više programa (time i više procesa) kako bi se maksimizirala produktivnost procesora. Određeni procesi povezani su s korisnikom (on ih je pokrenuo ili komuniciraju s njim) dok se ostali izvode u pozadini i odrađuju specifične funkcije (*daemon*) (Tanenbaum, 2015). S obzirom da procesor računala u jednom trenutku može izvoditi samo jedan proces, dojam istovremenog izvođenja stječe se pseudoparalelnim izvođenjem programa, tj. cikličkim izvođenjem svakog procesa određeni broj milisekundi. U trenutku kada procesor stane izvoditi proces i prijeđe na sljedeći, svi se bitni podaci o njemu (stanje, programski brojač, registri procesora, informacije o redoslijedu izvođenja, informacije o adresnom prostoru u radnoj memoriji i o adresnom prostoru u vanjskoj memoriji) spremaju u tabelu procesa (naziva se i kontrolni blok procesa) operacijskog sustava, kako bi se proces pri sljedećem dodjeljivanju procesora mogao nastaviti izvoditi. Proces se može nalaziti u stanju spremnosti, stanju izvođenja ili stanju blokiranosti (stanje u kojem proces čeka da se ispune uvjeti važni za nastavak njegovog izvođenja). Upravljač procesorom pomoću algoritma za dodjelu prioriteta određuje redoslijed izvođenja procesa. Algoritmi za dodjelu prioriteta procesima koji se koriste u operacijskim sustavima za interaktivne sustave su: Round-Robin, Priority algoritam, Multiple queues, Shortest process next, Guaranted scheduling, Lottery scheduling i Fair-Share scheduling (Kovačić, 2008).

Procesi su jedinstveno identificirani PID kodom i nastaju kod inicijalizacije sustava sistemskim pozivom procesa koji se trenutno izvodi, korisničkim zahtjevom ili pokretanjem serijske obrade. Proces koji se već izvodi pokreće stvaranje novog procesa slanjem sistemskog poziva operacijskom sustavu. U operacijskim sustavima koji se temelje na UNIX-u postoji hijerarhija među procesima. Pri pokretanju operacijskog sustava, pokreće se poseban proces nazvan init ili systemd (kojem je dodijeljen PID 1), koji stvara novi proces za svaki terminal u operacijskom sustavu. Stvoreni procesi čekaju prijavu korisnika u terminal te mogu stvarati nove procese. Proces koji je pozvao sistemski poziv za stvaranje novog procesa te novostvoreni proces u hijerarhijskoj su vezi roditelj-dijete. Svaki proces (osim procesa init i određenih daemon procesa) ima roditelja, te svi procesi zajedno čine hijerarhijsko stablo s procesom init u korijenu

(*root*). Kod operacijskog sustava Windows ne postoji hijerarhija među procesima već samo veza roditelj – dijete, gdje proces roditelj ima ovlasti (*token*) nad procesom djetetom i može ih predati drugom procesu. U operacijskim sustavima UNIX, za stvaranje novog procesa koristi se sistemski poziv fork dok se u Windows operacijskim sustavima koristi Win32 funkcija CreateProcess. Nakon uspješnog izvođenja, proces poziva sistemski poziv (exit u UNIX-u) za završetak izvođenja. U slučaju da je prilikom izvođenja došlo do greške uzrokovane procesom ili greške neovisne o njemu, proces se prekida. Do prekida izvođenja procesa može doći i u slučaju da drugi proces pozove sistemski poziv (kill u UNIX operacijskim sustavima) kojim ga prekida (Tanenbaum, 2015).

Još jedan pojam vezan uz proces, je procesna nit. Procesna nit može se definirati kao osnovna jedinica korištenja procesora ili kao proces u procesu. Svaki proces ima barem jednu procesnu nit a može ih imati i više. Procesne niti dijele sve resurse procesa i njihovo pokretanje je jednostavnije od pokretanja novog procesa. Pri stvaranju novog procesa potrebno je kopirati cijeli sadržaj memorije procesa dok je kod stvaranja niti dovoljno kopirati stog (Tanenbaum, 2015).

3.1 Funkcije vezane za trenutačni proces i korisnika

Opisi funkcionalnosti i sintakse programskog jezika Python u ovome potpoglavlju rada izrađeni su na temelju sljedećih izvora: (Miletić, 2021), (Hellman, 2011) i (Python Software Foundation, os — Miscellaneous operating system interfaces, 2021).

Svaki korisnik koji se prijavljuje u operacijski sustav ima jedinstveni identifikacijski kod (UID) pomoću kojeg ga operacijski sustav prepoznaje i provjerava njegove ovlasti. Svaki operacijski sustav ima jednog korisnika s većim ovlastima. U Linux operacijskim sustavima taj se korisnik naziva *Superuser* (UID 0), dok se u Windows operacijskim sustavima naziva Administrator (Tanenbaum, 2015). UID korisnika dijeli se na efektivni (EUID), stvarni (UID) i spremljeni (SUID). EUID određuje korisnikove ovlasti kod slanja signala procesima, stvarni UID utječe na korisnikovu mogućnost stvaranja i pristupanja datotekama a SUID se koristi kao mjesto za spremanje starog EUID-a kad se korisniku promijeni EUID i smanje mu se ovlasti za slanje signala procesima (Real, Effective and Saved UserID in Linux, 2021). Korisnici mogu biti grupirani u grupe te imati grupni identifikacijski kod (GID). Grupiranje korisnika ne radi

samostalno operacijski sustav, već administrator (*superuser*) sustava. GID se isto dijeli na efektivni, stvarni i spremljeni. Kad korisnik stvori novu datoteku ili pokrene proces, oni su označeni njegovim UID-om i GID-om grupe (ako korisnik pripada određenoj grupi). Stvorenoj datoteci dodjeljuje se i niz dozvola kojima se regulira pristup (čitanje, pisanje i pokretanje) za svaku klasu (korisnik, grupa i ostali) (Tanenbaum, 2015). Dozvole se često zapisuju oktalnom notacijom tako da se svakoj dozvoli dodijeli težinska vrijednost (4 za čitanje, 2 za pisanje i 1 za pokretanje) i vrijednosti se zasebno zbroje za svaku klasu. Ako za stvorenu datoteku sve klase imaju sve dozvole, oktalnom notacijom to se zapisuje kao 777, a u slučaju da niti jedna klasa nema niti jednu dozvolu, zapis je 000. Na primjeru datoteke sa oktalnim zapisom 631 moguće je uočiti dozvole za pojedinu klasu. Prvi broj označava dozvole korisnika, drugi broj označava dozvole grupe a posljednji broj označava dozvole ostalih korisnika. Dozvole korisnika označene su brojem 6: to je zbroj dozvole za čitanje (vrijednost 4) i dozvole za pisanje (vrijednost 2). Grupa korisnika može pisati (vrijednost 2) i pokretati datoteku (vrijednost 1) dok ostali mogu samo pokretati datoteku (vrijednost 1).

Korištenjem određenih funkcija modula os možemo pročitati i mijenjati identifikacijski kod procesa i korisnika te mijenjati prioritet izvođenja procesa.

U sljedećem primjeru (Slika 2), pomoću funkcija modula os vezanih za trenutačni proces i korisnika, prikazan je program koji ispisuje informacije operacijskog sustava (`os.uname()`), ime prijavljenog korisnika (`os.login()`), te identifikacijske kodove korisnika (`getuid()`, `geteuid()`), grupe kojoj korisnik pripada (`getegid()`, `getgid()`), dodatnih grupa kojima korisnik pripada uz primarnu grupu (funkcija `getgroups()` koja vraća niz GID kodova vezanih uz trenutačni proces) i kodova vezanih za proces (`getpid()`, `getppid()`, `getpgid(pid procesa)` i `getsid(pid procesa)`). Dohvaćanje specifičnih informacija operacijskog sustava moguće je referenciranjem atributa funkcije `uname()`: atributom `sysname` dohvaća se naziv operacijskog sustava, atributom `nodename` dohvaća se naziv računala povezanog na mrežu, atributom `release` specificirano je izdanje operacijskog sustava, atributom `version` definirana je inačica operacijskog sustava i atributom `machine` dohvaćaju se podaci o hardveru računala. Primjerice, za dohvaćanje izdanja operacijskog sustava koristi se funkcija `os.uname().release`. Prikazan je i dodijeljeni korisnički prioritet izvođenja (*niceness*) za korisnika, proces i grupu procesa (`getpriority(vrsta, identifikacijski kod)`).

Modul os nudi različite funkcije vezane uz promjenu UID-a korisnika i grupe: funkcija `setuid(vrijednost)` mijenja stvarni i efektivni UID na zadanu vrijednost, funkcija `seteuid(vrijednost)` mijenja samo efektivni UID, dok funkcija `setreuid(UID, EUID)` nudi mogućnost promjene stvarnog i efektivnog UID-a na različite vrijednosti. Takve funkcije postoje i za promjenu GID-a. Ako korisnik koji nema dopuštenje (nije *superuser*) pokuša promijeniti UID, sustav vraća grešku zbog nedostatka ovlaštenja za tu akciju. Važno je napomenuti da operacijski sustavi imaju određena pravila pri dodjeljivanju identifikacijskih kodova. U Linux operacijskom sustavu identifikacijski kodovi su brojčane vrijednosti i prijavljenim korisnicima dodjeljuje se broj od tisuću do 59999 (The Operating System, 2021).

Funkcijom `getpriority` provjerava se prioritet izvođenja dodijeljenog za proces, korisnika ili grupu procesa. Prvim argumentom određuje se vrsta prioriteta (za proces, korisnika ili grupu procesa) a drugi argument služi za identifikaciju pomoću identifikacijskog koda. U primjeru su za prvi argument korištene varijable `PRIO_PROCESS`, `PRIO_PGRP` i `PRIO_USER` sadržane u modulu os, dok je za drugi argument korišten broj nula (0) kojim se identificira trenutačni proces, korisnik ili grupa procesa.

Korisnička maska služi za ograničavanje dozvola dodijeljenih datoteci pri kreiranju (File security, 2021) i mijenja se funkcijom `umask(oktalni zapis)`. Važno je uočiti kako se funkcijom `umask` ne dodaju dozvole već ograničenja. Funkcija prima jedan argument, oktalni zapis dozvola koje se želi oduzeti od zadane (*default*) vrijednosti i vraća zbroj vrijednosti prijašnje korisničke maske. Oktalni zapis u programskom jeziku Python označava se oznakom „0o“ ispred zapisa (binarni zapis označava se oznakom „0b“ ispred zapisa). Datoteke i direktoriji razlikuju se po dozvolama zadanim kod stvaranja. Datoteka stvorena uz korisničku masku vrijednosti 000 imat će vrijednost 666 kao zapis dozvola, dok će direktorij imati zapis 777. Promjenom korisničke maske na vrijednost, primjerice 026, datoteke će pri stvaranju imati dozvole zapisane u obliku 640 a dozvole direktorija bit će 751.

Moguće je i utjecati na prioritet izvođenja procesa, korisnika ili grupe procesa korištenjem funkcije `setpriority()`. Prvi i drugi argument funkcije isti su kao i za funkciju `getpriority()`, dok se trećim argumentom zadaje vrijednost koja će se pridodati prioritetu kojeg zadaje sustav. Ukupni prioritet izvođenja procesa određuje se zbrajanjem zadanog prioriteta sustava s prioritetom kojeg je odredio korisnik. Prioritet izvođenja kojeg je odredio korisnik još se naziva i *niceness*. Zadana vrijednost sustavnog prioriteta najčešće je 20 i njemu se pridodaje vrijednost *nicenessa* koja se može kretati od -20 do 19. Stoga, vrijednost ukupnog

prioriteta može biti od 0 do 39: što je vrijednost manja prioritet je veći (Slavuj, 2021). Dozvolu za korištenje bilo koje spomenute set funkcije ima samo korisnik s efektivnim UID-om nula (*superuser*). U primjeru se koristi funkcija `strerror(broj greške)` kojoj se prosljeđuje broj greške i ona vraća poruku s objašnjenjem greške. Ako set funkcije upotrijebi korisnik bez dozvole, javlja se greška 13 koja označava odbijanje dozvole.

```
import os

def info():
    print("Informacije sustava, korisnika i procesa:\nos: %s \nlogin: %s"\
        %(os.uname(), os.getlogin()))

def greska():
    print("%s - Nije moguce primijeniti promjene jer korisnik nije root"\
         %(os.strerror(13)))

def prioritet():
    print("\nprioritet\nkorisnika: %s procesa: %s grupe procesa: %s\n" \
        %(os.getpriority(os.PRIO_USER,0), os.getpriority(os.PRIO_PROCESS,0),\
             os.getpriority(os.PRIO_PGRP,0)))

    if os.geteuid() != 0:
        greska()

    else:
        print("promjena prioriteta zadanog od strane korisnika")
        os.setpriority(os.PRIO_USER,0,-15)
        os.setpriority(os.PRIO_PROCESS,0,-2)
        print("procesa: %s korisnika: %s"\
             %(os.getpriority(os.PRIO_PROCESS,0), os.getprior-
ity(os.PRIO_USER,0)))

def id_procesa():
    print("\nid roditelja procesa: %s id procesa: %s id grupe procesa: %s \
        id sesije procesa: %s" %(os.getppid(), os.getpid(),\
             os.getpgid(os.getpid()), os.getsid(os.getpid())))

def id_grupe():
    print("\ngrupe kojima korisnik pripada: %s" %(os.getgroups()))
    print("GID grupe: \nstvarni GID: %s efektivni GID: %s"\
         %(os.getgid(), os.getegid()))

    if os.geteuid() != 0:
        greska()
```

```
    else:
        print("promjena stvarnog GID-a")
        os.setregid(200,0)
        print("stvarni GID: %s efektivni GID: %s" %(os.get-
gid(), os.getegid()))
        print("promjena efektivnog i stvarnog GID-a na istu vrijednost:")
        os.setgid(300)
        print("stvarni GID: %s efektivni GID: %s" %(os.get-
gid(), os.getegid()))

def id_korisnika():
    print("\nUID korisnika:\nstvarni uid: %s efektivni uid: %s "\
         %(os.getuid(), os.geteuid()))

    if os.geteuid() != 0:
        greska()

    else:
        print("Promjena stvarnog UID-a")
        os.setreuid(500,0)
        print("efektivni uid: %s stvarni uid: %s" \
%(os.geteuid(),os.getuid()))
        print("Promjena efektivnog UID-a")
        os.seteuid(600)
        print("efetivni uid: %s stvarni uid: %s" \
%(os.geteuid(), os.getuid()))

def promjena_maske():
    print("vrijednost prijasnje korisnicke maske: %s" \
%(os.umask(0o002)))


info()
prioritet()
id_procesa()
id_grupe()
id_korisnika()
promjena_maske()
```

Slika 2. Primjer rada s funkcijama vezanim za trenutačni proces i korisnika

Slijedi izlaz programa pokrenutog od strane *superusera*:

```
[marko@localhost zavrsni_python]$ sudo python procesi.py

Informacije sustava, korisnika i procesa:

os: posix.uname_result(sysname='Linux', nodename='localhost.localdomain',
release='5.11.10-200.fc33.x86_64', version='#1 SMP Thu Mar 25 16:51:31 UTC 2021',
machine='x86_64')

login: marko

prioritet

korisnika: -15 procesa: 0 grupe procesa: 0



promjena prioriteta zadanog od strane korisnika

procesa: -2 korisnika: -15

id roditelja procesa: 12646 id procesa: 12659 id grupe procesa: 12646 id sesije
procesa: 6946



grupe kojima korisnik pripada: [0]

GID grupe:

stvarni GID: 0 efektivni GID: 0

promjena stvarnog GID-a

stvarni GID: 200 efektivni GID: 0

promjena efektivnog i stvarnog GID-a na istu vrijednost:

stvarni GID: 300 efektivni GID: 300



UID korisnika:

stvarni uid: 0 efektivni uid: 0

Promjena stvarnog UID-a

efektivni uid: 0 stvarni uid: 500

Promjena efektivnog UID-a

efektivni uid: 600 stvarni uid: 500

vrijednost prijasnje korisnicke maske: 18
```

Slijedi izlaz programa pokrenutog od strane običnog korisnika:

```
[marko@localhost zavrsni_python]$ python procesi.py

Informacije sustava, korisnika i procesa:

os: posix.uname_result(sysname='Linux', nodename='localhost.localdomain',
release='5.11.10-200.fc33.x86_64', version='#1 SMP Thu Mar 25 16:51:31 UTC 2021',
machine='x86_64')

login: marko

prioritet

korisnika: -11 procesa: 0 grupe procesa: 0



Permission denied - Nije moguce primijeniti promjene jer korisnik nije root



id roditelja procesa: 6946 id procesa: 12713 id grupe procesa: 12713 id sesije
procesa: 6946



grupe kojima korisnik pripada: [10, 977, 1000]

GID grupe:

stvarni GID: 1000 efektivni GID: 1000

Permission denied - Nije moguce primijeniti promjene jer korisnik nije root



UID korisnika:

stvarni uid: 1000 efektivni uid: 1000

Permission denied - Nije moguce primijeniti promjene jer korisnik nije root

vrijednost prijasnje korisnicke maske: 2
```

3.2 Funkcije za stvaranje i upravljanje procesima

Opisi funkcionalnosti i sintakse programskog jezika Python u ovome potpoglavlju rada izrađeni su na temelju sljedećih izvora: (Miletić, 2021), (Hellman, 2011) i (Python Software Foundation, os — Miscellaneous operating system interfaces, 2021).

Program prikazan u sljedećem primjeru (Slika 3) koristi funkcije modula os za stvaranje procesa i njihovo upravljanje.

Funkcijom `os.fork()` implementiran je sistemski poziv za stvaranje novog procesa. Funkcija `fork` procesu roditelju vraća PID procesa djeteta, dok djetetu vraća vrijednost 0, što pomaže pri njihovom prepoznavanju. Stvoreni proces dijete je klon roditelja te će izvoditi iste naredbe u programu osim dijelova uvjetovanih PID-om. U primjeru se funkcijom `fork()` stvara novi proces te se njezina povratna vrijednost (PID djeteta kod procesa roditelja, te 0 kod djeteta) sprema u varijablu „`pid`“. U procesu roditelju se izvodi funkcija `os.wait()`, kojom proces čeka završetak izvođenja procesa djeteta. `Wait` vraća n-torku u kojoj je prva vrijednost PID djeteta a druga vrijednost izlazni status. Vraćena n-torka sprema se u varijablu „`cekanje`“ i njezini elementi dohvaćaju se pojedinačno referenciranjem njihovog indeksa. Važno je koristiti funkciju `wait` kako bi proces roditelj pročitao izlazni status procesa djeteta jer proces dijete neće završiti izvođenje (nalazit će se u suspendiranom stanju) dok njegov izlazni status nije pročitan (Tanenbaum, 2015). Proces dijete ispisati će svoj PID i PID roditelja, pomoću funkcije `times()` ispisat će trenutačna vremena procesa (vrijeme korisnika, sustava, svih procesa djece korisnika, svih procesa djece sustava i proteklo vrijeme od određenog fiksnog događaja).

Korištenjem funkcije `sched_getaffinity(PID)` proces dijete provjerava na koje je jezgre procesora ograničen proces roditelj. `Sched_getaffinity` kao argument prima PID procesa za kojeg se želi izvesti provjera. Nakon toga prikazana je upotreba funkcije `system(naredba)`. Funkciji `system` se argumentom prosljeđuje naredba koja se izvodi u terminalu operacijskog sustava. U primjeru je korištena naredba date koja ispisuje vrijeme i datum.

Još jedna od često korištenih funkcija je `nice(nova vrijednost` *`nicenessa`*`)` kojom se mijenja vrijednost varijable *niceness* procesa. *Niceness* procesa je već spomenuta vrijednost na temelju koje se određuje ukupni prioritet izvođenja procesa. Zadani prioritet i *niceness* ne

smiju se poistovjećivati: prioritet određuje sustav na temelju *nicenessa* kojeg odredi korisnik, najčešće pribrojavanjem *nicenessa* zadanome prioritetu (Slavuj, 2021). Potrebno je napomenuti da funkcijom `nice` samo tzv. *superuser* može smanjivati vrijednost *nicenessa*, dok ju običan korisnik može samo povećavati. Nakon promjene *nicenessa* proces dijete, funkcijom `_exit(status)`, završava izvođenje. `_exit(status)` kao argument prima izlazni status procesa. Preporučeno je da se `_exit` koristi kod procesa djeteta nakon funkcije `fork` a u svim ostalim slučajevima da se koristi `exit` funkcija modula `system`. Nakon što je proces dijete završio s izvođenjem, proces roditelj prekida izvođenje funkcijom `abort()`. `Abort` generira signal `SIGABRT` kojim se prekida izvođenje trenutnog procesa.

```
import os

pid = os.fork()

if pid>0:
    print("Izvodi se proces roditelj %s" %(os.getpid()))
    print("cekanje zavrsetka izvodenja procesa djeteta")
    cekanje = os.wait()
    print("\nproces dijete %s je zavrsilo izvodenje s izlaznim statusom %s"\
        %(cekanje[0], cekanje[1]))
    print("prekid izvođenja procesa roditelja")
    os.abort()

else:
    print("\nIzvodi se proces dijete")
    print("proces dijete %s i roditelj %s" %(os.getpid(), os.getppid()))
    print(os.times())
    print("Jezgre kojima je dodjeljen proces roditelj: %s"\
        %(os.sched_getaffinity(os.getppid())))
    naredba = 'date'
    os.system(naredba)
    prioritet = os.getpriority(os.PRIO_PROCESS,0)
    os.nice(-5)
    print("promjena prioriteta procesa pomocu nicenessa s %s na %s"\
        %(prioritet, os.getpriority(os.PRIO_PROCESS,0)))
    os._exit(0)
```

Slika 3. Primjer rada s funkcijama za stvaranje i upravljanje procesima

Izlaz programa pokrenutog s ovlastima *superusera*:

```
[marko@localhost zavrsni_python]$ sudo python stvaranje_procesa.py

Izvodi se proces roditelj 12369

cekanje zavrsetka izvodenja procesa djeteta


Izvodi se proces dijete

proces dijete 12370 i roditelj 12369

posix.times_result(user=0.0, system=0.0, children_user=0.0, children_system=0.0,
elapsed=4330442.68)

Jezgre kojima je dodjeljen proces roditelj: {0, 1, 2, 3, 4, 5, 6, 7}

ned, 22.08.2021.  18:30:28 CEST

promjena prioriteta procesa pomocu nicenessa s 0 na -5


proces dijete 12370 je zavrsilo izvodenje s izlaznim statusom 0

zavrsetak izvođenja procesa roditelja

Prekid
```

3.3 Stvaranje deamon procesa

Opisi funkcionalnosti i sintakse programskog jezika Python u ovome potpoglavlju rada izrađeni su na temelju sljedećih izvora: (Miletić, 2021), (Hellman, 2011) i (Python Software Foundation, os — Miscellaneous operating system interfaces, 2021).

U nastavku (Slika 4) je prikazan primjer stvaranja jednostavnog *daemon* procesa. *Daemon* proces je proces pokrenut u pozadini koji nije povezan niti s jednim terminalom (Miletić, 2021). Primjeri nekih od poznatijih *daemon* procesa su sshd (pruža kriptiranu komunikaciju preko mreže), syslogd (bilježi poruke aplikativnih i sistemskih procesa) i httpd (odgovara na serverske zahtjeve). Često se kao njegov roditelj vodi proces init (s PID-om 1) i nastaje dvostrukim korištenjem funkcije `fork()`. Proces izvodi funkciju `fork()` te završava izvođenje. Pritom proces dijete funkcijom `chdir(putanja direktorija)` mijenja radni direktorij procesa u direktorij `/`, funkcijom `umask(vrijednost maske)` mijenja korisničku masku na 0 i funkcijom `setsid()` postaje voditelj sesije čime postiže odvajanje od terminala. Nakon što

je ispunio sve uvjete za nastanak *daemon* procesa, izvodi funkciju `fork()` i nakon stvaranja novog procesa završava izvođenje. Kreirani proces je *daemon* proces i ispisuje PID roditelja (procesa init).

```
import os

pid = os.fork()

if pid > 0:
    os._exit(0)
else:
    os.chdir('/')
    os.setsid()
    os.umask(0)
    print("radni direktorij procesa: %s" %(os.getcwd()))
    print("PID procesa: %s" %(os.getpid()))
    print("PID voditelja sesije: %s" %(os.getsid(os.getpid())))
    pid2 = os.fork()

    if pid2 > 0:
        os._exit(0)
    else:
        print("pokrenut daemon proces")
        print("PID roditelja daemon procesa: %s" %(os.getppid()))
        print("daemon proces zavrsen")
        os._exit(0)
```

Slika 4. Primjer stvaranja *daemon* procesa funkcijama modula os

Slijedi izlaz programa pokrenutog od strane *superusera*:

```
[marko@localhost zavrsni_python]$ sudo python daemon.py
radni direktorij procesa: /
PID procesa: 5247
PID voditelja sesije: 5247
pokrenut daemon proces
PID roditelja daemon procesa: 1
PID procesa: 5248
voditelj sesije: 1
daemon proces zavrsen
```

3.4 Slanje signala procesu

Opisi funkcionalnosti i sintakse programskog jezika Python u ovome potpoglavlju rada izrađeni su na temelju sljedećih izvora: (Miletić, 2021), (Hellman, 2011) i (Python Software Foundation, os — Miscellaneous operating system interfaces, 2021).

Signali služe za međusobnu komunikaciju procesa ili za obavještavanje procesa o određenom događaju, a definirani su nazivom i rednim brojem. Pri slanju signala procesu se prosljeđuje redni broj željenog signala. Nakon primitka signala proces zaustavlja izvođenje i rukuje signalom (Tanenbaum, 2015). Proces može rukovati signalom na različite načine, ovisno o primljenom signalu: može završiti izvođenje, zaustaviti izvođenje, ignorirati signal ili nastaviti izvođenje nakon zaustavljanja. Signali mogu biti sinkroni ili asinkroni. Sinkroni signali nastaju pri izvođenju procesa dok asinkrone signale pokreću operacijski sustav ili aplikativni programi. Neki od najčešće korištenih signala su `SIGKILL(9)` (prekida izvođenje procesa), `SIGSTOP(19)` (zaustavlja izvođenje procesa), `SIGCONT(18)` (nastavlja izvođenje procesa) i `SIGINT(2)` (pokreće prekid izvođenja procesa pomoću tipkovnice). Proces ne može ignorirati niti rukovati signalima `SIGKILL` i `SIGSTOP` (Slavuj, 2021).

U sljedećem primjeru (Slika 5) prikazano je korištenje funkcije `kill(PID, signal)` kojom proces šalje signale drugom procesu. Korištena je i funkcija `sleep(n)` iz modula `time`, kojom se zaustavlja izvođenje procesa za vrijeme n, kako bi se lakše uočilo prekidanje izvođenja procesa djeteta. Funkciji `kill` prosljeđuje se PID procesa kojem se šalje signal i određeni signal iz modula `signal` koji se želi poslati. U primjeru proces stvara dijete, čeka 5 sekundi te prekida izvođenje djeteta signalom `SIGKILL`. U isto vrijeme, dijete se izvodi i u intervalu od jedne sekunde ispisuje poruku. Da proces roditelj, nakon 5 sekundi, ne prekine izvođenje procesa djeteta, dijete bi nastavilo ispisivati poruke i ne bi prekinulo izvođenje. U slučaju da se funkciji kill proslijedi PID kod nepostojećeg procesa, javit će se greška 3 kojom se označava nepostojanje željenog procesa.

```python
import os,signal,time
pid = os.fork()
if pid > 0:
    time.sleep(5)
    os.kill(pid, signal.SIGKILL)
    print("prekinuto izvodenje procesa djeteta")
else:
    print("pokrenuto izvodenje procesa djeteta")
    for i in range(10):
        time.sleep(1)
        print("proces dijete se izvodi")
```

Slika 5. Primjer slanja signala procesu

Slijedi izlaz programa pokrenutog od strane *superusera*:

```
[marko@localhost zavrsni_python]$ sudo python kill.py

pokrenuto izvodenje procesa djeteta

proces dijete se izvodi

proces dijete se izvodi

proces dijete se izvodi

proces dijete se izvodi

prekinuto izvodenje procesa djeteta
```

4. Rad s datotekama i direktorijima

Operacijski sustav mora ispuniti određene uvjete vezane uz pohranu podataka, mora biti sposoban pohraniti veće količine podataka, podaci moraju ostati sačuvani (i u slučaju iznenadnog prekida izvođenja procesa) te moraju biti dohvatljivi od strane više različitih procesa istovremeno. Spomenuti uvjeti su ispunjeni uvođenjem koncepta datoteke (*file*) (Tanenbaum, 2015).

Datoteka je još jedan temeljni koncept operacijskog sustava: to je logička jedinica podataka koju stvara proces. Pri stvaranju, datoteci se definiraju naziv i svojstva i ona se pohranjuje u trajnu memoriju sustava sve dok ju korisnik sam ne izbriše. Na postojanje datoteke ne smije utjecati stvaranje, prekidanje ili završavanje izvođenja procesa (Tanenbaum, 2015). Pravila imenovanja datoteka variraju ovisno o operacijskom sustavu, ali kod većine sustava dopušteno je korištenje slova, brojeva i određenih posebnih znakova do duljine od 255 znakova. Naziv datoteke sastoji se od dva dijela razdvojenih točkom: prvi dio naziva određuje korisnik, dok drugi dio (ekstenzija) ovisi o programu koji izvodi datoteku (Kovačić, 2008).

Nad datotekama je moguće izvoditi razne operacije korištenjem sistemskih poziva: stvaranje, brisanje, otvaranje, zatvaranje, čitanje, pisanje, dodavanje, pretraživanje, preimenovanje te čitanje i postavljanje atributa. Njihovim kombiniranjem može se izvesti i razne druge operacije (Silberschatz, Galvin, & Gagne, 2008).

Svojstva datoteke određena su njezinim atributima. U važnije atribute datoteke spadaju vlasnik datoteke, tip, vrijeme stvaranja, vrijeme posljednje promjene sadržaja, vrijeme posljednjeg pristupa, pravo pristupa i veličina. Ostali atributi datoteke mogu se razlikovati ovisno o operacijskom sustavu (Tanenbaum, 2015).

Datoteke se mogu razlikovati po strukturi, načinu pristupa i tipu. Ovisno o strukturi, datoteke mogu imati strukturu bajta, strukturu zapisa i strukturu stabla. Operacijski sustavi koji se temelje na UNIX-u i Windows operacijski sustavi upotrebljavaju datoteke sa strukturom bajta zbog njihove fleksibilnosti (Tanenbaum, 2015). Prema načinu pristupa, datoteke se dijele na datoteke sa sekvencijalnim pristupom i datoteke sa nasumičnim (direktnim) pristupom (Silberschatz, Galvin, & Gagne, 2008). Operacijski sustav razlikuje više tipova datoteka. Regularne (ili obične) datoteke sadrže korisničke podatke i mogu biti ASCII (tekstualne) datoteke ili binarne. Direktorij je sistemski tip datoteke kojim se definira i održava struktura

datotečnog sustava. Postoje i posebne znakovne datoteke i posebne blokovske datoteke koje služe pri radu s ulazno-izlaznim uređajima (Kovačić, 2008).

Datotečnim sustavom računala upravlja dio operacijskog sustava koji se naziva upravljač datotečnog sustava (*file system*). Noviji Windows operacijski sustavi koriste NTFS upravljač datotečnog sustava, dok su stariji koristili FAT-16 i FAT-32 upravljače datotečnog sustava (Tanenbaum, 2015). Većina distribucija operacijskog sustava Linux koristi upravljač datotečnog sustava EXT4 (Both, 2021). Upravljači datotečnog sustava upravljaju datotečnim sustavom pomoću direktorija. Direktoriji su sistemske datoteke koje sadrže podatke o strukturi datotečnog sustava. Moderni datotečni sustavi imaju hijerarhijsku strukturu direktorija (stablo direktorija). U jednokorijenskim operacijskim sustavima postoji samo jedan korijenski direktorij (korijen) unutar kojeg se nalaze sve ostale datoteke i direktoriji (bez obzira na broj particija), dok u višekorijenskim operacijskim sustavima svaka particija ima jedan korijenski direktorij. Hijerarhijskom strukturom datotečnog sustava postiže se veća preglednost i bolja organizacija datoteka (Tanenbaum, 2015).

Za specificiranje lokacije određenog direktorija ili datoteke u datotečnom sustavu koriste se putanje (*paths*). Putanja direktorija može biti apsolutna ili relativna. Apsolutnom putanjom prikazan je put kroz direktorije kojim se treba proći kako bi se iz korijenskog direktorija (označenog s početnim „/“) došlo do željenog direktorija. Relativna putanja prikazuje put iz trenutnog radnog direktorija do odredišnog (Tanenbaum, 2015). Kod relativnih putanja često je potrebno vratiti se na višu razinu direktorija što se označava s dvije točke (`..`). Primjerice, „`/home/marko/zavrsni_rad`” je apsolutna putanja kojom se iz korijenskog direktorija dolazi do direktorija „`zavrsni rad`” u kojem se nalazi ovaj završni rad, a „`../../marko/zavrsni_rad`” je primjer relativne putanje kojom se iz direktorija „`primjer direktorija`”, koji se nalazi na radnoj površini unutar kućnog direktorija korisnika, dolazi do direktorija „`zavrsni rad`”.

Određene operacije dostupne pri radu s regularnim datotekama (stvaranje, brisanje, otvaranje, čitanje, pisanje/izmjena i preimenovanje) dostupne su i pri radu s direktorijima. Dvije operacije specifične za direktorije su povezivanje direktorija i prekidanje veza između direktorija. Povezivanjem se jedna datoteka može prikazivati u više različitih direktorija. Veza (*link*) može biti čvrsta ili simbolička. Čvrstom vezom definirana se datoteka veže uz ime u željenoj putanji, čime se postiže da više datoteka različitih imena pokazuje na isti skup podataka u memoriji.

Simboličkom vezom stvorena datoteka pokazuje na već postojeću datoteku, koja pokazuje na skup podataka u memoriji (Tanenbaum, 2015).

4.1 Funkcije za upravljanje datotekama i direktorijima

Opisi funkcionalnosti i sintakse programskog jezika Python u ovome potpoglavlju rada izrađeni su na temelju sljedećih izvora: (Miletić, 2021), (Hellman, 2011) i (Python Software Foundation, os — Miscellaneous operating system interfaces, 2021).

Primjerom u nastavku (Slika 6) prikazano je izvođenje operacija nad datotekama i direktorijima pomoću funkcija iz modula os. Za određene operacije korištene su i funkcije iz modula os.path koji je namijenjen za upravljanje putanjama datoteka.

U varijablu „`aps_putanja`", pomoću funkcije `getcwd()`, spremljena je apsolutna putanja trenutnog radnog direktorija u kojem se izvodi program. Program ispisuje spomenutu apsolutnu putanju te sam naziv direktorija u kojem se program izvodi.

Za ispis sadržaja trenutnog direktorija korištena je funkcija `os.path.basename(putanja)` modula `os.path`. Funkcija `basename` kao argument prima putanju na temelju koje određuje osnovno ime direktorija. Nakon toga, program ispisuje sve datoteke i direktorije koji su sadržani u radnom direktoriju. To se postiže funkcijom `listdir(putanja)` kojoj je proslijeđena varijabla s apsolutnom putanjom direktorija.

Nakon ispisa, od korisnika se traži da upiše apsolutnu putanju s direktorijem kojeg želi kreirati. Kreiranje direktorija moguće je funkcijama `mkdir(putanja, mode=0o777)` i `makedirs(putanja, mode=0o777, exist_ok=False)`. U ovom primjeru korištena je funkcija `makedirs`, jer ona omogućuje rekurzivno kreiranje svih direktorija koji ne postoje u navedenoj putanji. Suprotno tome, funkcija `mkdir` kreira samo završni direktorij u putanji i u slučaju da navedena putanja ne postoji vraća grešku. `Makedirs` prima željenu putanju kao obvezan argument a argumente `mode` i `exist_ok` kao opcionalne. Argumentom `mode` (način pristupa) određuju se dozvole (čitanje, pisanje i izvođenje) direktorija, dok se argumentom `exist_ok` određuje akcija koja će se poduzeti u slučaju da direktorij već postoji. Ako je `exist_ok` postavljen na vrijednost `false`, a direktorij već postoji, funkcija vraća grešku. Ako je postavljen na `true`, funkcija ne vraća ništa. Metoda `mkdir` kao argumente

prima putanju i način pristupa definirane kao i kod funkcije `makedirs`. Nakon što su funkcijom `makedirs` stvoreni novi direktoriji, program koristi funkciju `chdir(putanja)` kojom se mijenja radni direktorij programa.

Kao što je prikazano programom, za prikaz relativne putanje između dva direktorija potrebno je koristiti funkciju `os.path.relpath(putanja,start=os.curdir)`. Funkciji `relpath` proslijeđena su dva argumenta: apsolutna putanja traženog direktorija i početni direktorij za relativnu putanju argumentom (`start`). Ako se želi saznati relativna putanja s početkom u trenutnom direktoriju, argument `start` se postavlja na vrijednost varijable `os.curdir` (varijabla `os.curdir` služi operacijskom sustavu za označavanje trenutnog direktorija). U primjeru se kao rezultat funkcije `relpath` ispisuje relativna putanja „`../../../zavrsni_python`“ kojom se iz tek stvorenog direktorija „`/home/marko/Dokumenti/zavrsni/primjer`“ može doći do direktorija s apsolutnom putanjom „`/home/marko/zavrsni_python`“.

Za stvaranje nove datoteke „`test.txt`” upotrjebljena je funkcija `open(naziv,način)`. `Open` je jedna od predefiniranih funkcija jezika Python koju nije potrebno uvoziti (`input`, `print` i `format` su još neke često korištene predefinirane funkcije jezika Python). Funkciji `open` prosljeđuje se naziv datoteke i način korištenja (pisanje - 'w', čitanje - 'r', kreiranje - 'x', dodavanje - 'a', binarni način - 'b', tekstualni način - 't' i ažuriranje - '+'). Kad je datoteka stvorena, u primjeru se u nju upisuju prva tri reda i njezina se apsolutna putanja sprema u varijablu „`putanja_datoteke`” koristeći funkciju `os.path.abspath(putanja)`. `Abspath` može primiti apsolutnu ili relativnu putanju kao argument.

Dohvaćanje raznih podataka o datoteci (poput raznih vremenskih oznaka, veličine, UID vlasnika i sl.) omogućeno je funkcijom `stat(putanja)` i njenim atributima. U ovom primjeru korišten je atribut `st_size` funkcije `stat` kako bi se provjerila veličina datoteke u bajtovima. Uz atribut `st_size` funkcija `stat` sadrži i atribute: `st_mode` (tip datoteke i način pristupa), `st_ino` (jedinstveni identifikator datoteke), `st_dev` (identifikator uređaja na kojem je datoteka spremljena), `st_nlink` (broj čvrstih poveznica datoteke), `st_uid` (UID vlasnika datoteke), `st_gid` (GID vlasnika datoteke), `st_atime` (vrijeme posljednjeg pristupa datoteci izraženo u sekundama), `st_mtime` (vrijeme posljednje promjene sadržaja datoteke izraženo u sekundama), `st_ctime` (u Windows operacijskim sustavima označava vrijeme stvaranja datoteke izraženo u sekundama, u operacijskim sustavima sličnim UNIX-u

označava vrijeme posljednje promjene meta podataka datoteke izraženo u sekundama), `st_atime_ns` (vrijeme posljednjeg pristupa datoteci izraženo u nanosekundama kao cijeli broj), `st_mtime_ns` (vrijeme posljednje promjene sadržaja datoteke izraženo u nanosekundama kao cijeli broj), `st_ctime_ns` (u Windows operacijskim sustavima označava vrijeme stvaranja datoteke izraženo u nanosekundama kao cijeli broj, u operacijskim sustavima sličnim UNIX-u označava vrijeme posljednje promjene meta podataka datoteke izraženo u nanosekundama kao cijeli broj). Za smanjenje veličine na željenu vrijednost, modul os nudi funkciju `truncate(putanja, veličina)` i u primjeru je prikazano smanjivanje datoteke s 28 bajtova na 14 (čitanjem datoteke nakon završetka izvođenja programa može se uočiti da više nisu upisana prva tri reda datoteke već manje, ovisno o smanjenju datoteke).

Vremenske oznake posljednjeg pristupa i posljednje promjene sadržaja datoteke moguće je izmijeniti funkcijom `utime(putanja, ntorka)`. Program najprije ispisuje spomenute vremenske oznake (atributi `st_atime` i `st_mtime` funkcije `stat`) pa korisniku nudi mogućnost promjene na željenu vrijednost koristeći funkciju `utime`. Funkciji `utime` potrebno je proslijediti putanju datoteke i n-torku s vrijednostima željenih vremenskih oznaka.

Iako je atributom `st_mode` funkcije `stat` moguće provjeriti dozvole datoteke, intuitivnije je koristiti funkciju `access(putanja, vrijednost dozvole)`. `Access` provjerava postojanje (prosljeđivanjem varijable `os.F_OK`), mogućnost čitanja (prosljeđivanjem varijable `os.R_OK`), mogućnost pisanja (prosljeđivanjem varijable `os.W_OK`) i mogućnost izvođenja (prosljeđivanjem varijable `os.X_OK`) datoteke proslijeđene putanjom.

Za promjenu zadanih dozvola datoteke, os modul nudi funkciju `chmod(putanja, nova dozvola)`, kojoj se određuje putanja datoteke i željene dozvole. U primjeru, program provjerava dozvole datoteke `test.txt` funkcijom `access`, sprema ih u varijable „`postoji1`”, „`citanje1`”, „`pisanje1`” i „`izvodenje1`” radi kasnije usporedbe, oduzima dozvolu za pisanje korisniku i ispisuje dozvole datoteke prije i poslije promjene. Važno je napomenuti da funkcija `chmod` kao argument dozvole prima i vrijednosti iz modula `stat` (u primjeru je korištena vrijednost `stat.S_IRUSR` kojom se označava dozvola čitanja za korisnika).

U primjeru je prikazano i stvaranje čvrste i simboličke poveznice na datoteku. Čvrsta poveznica kreira se funkcijom `link(putanja datoteke, željena putanja)`. Funkciji se kroz argumente specificira putanja datoteke te putanja poveznice (uključujući i ime poveznice) koju

se kreira. Za kreiranje simboličke poveznice koristi se funkcija `symlink(putanja datoteke, željena putanja)` koja prima iste argumente kao i funkcija `link`. U primjeru je prikazano stvaranje čvrste poveznice „`test(cvrsta_poveznica).txt`" i simboličke poveznice „`test(poveznica).txt`" na datoteku „`test.txt`".

Još jedna korisna funkcija modula `os` je `walk(putanja, topdown)`. Funkcija `walk` izvodi „šetnju" kroz direktorij naveden putanjom i svim njegovim poddirektorijima. Za svaki direktorij kroz koji prođe, funkcija vraća putanju, sadržane poddirektorije i datoteke. Funkcija `walk` prima i argument `topdown` kojim se specificira redoslijed prolaženja kroz direktorije (od vrha prema dnu ili obrnuto).

U primjeru su još prikazane i funkcije za preimenovanje jednog direktorija ili datoteke (`rename(stari naziv, novi naziv)`) ili više direktorija rekurzivno (`renames(stari nazivi, novi nazivi)`). Ako datoteka ili direktorij koji se želi preimenovati ne postoji, javit će se greška 2 koja definira nepostojanje datoteke ili direktorija. Program će javiti grešku i u slučaju da datoteka s novim nazivom već postoji. Ako se direktorij želi preimenovati nazivom drugog već postojećeg direktorija sustav će provjeriti već postojeći direktorij: ako je on prazan, biti će zamijenjen željenim direktorijem. U protivnom javit će se greška 17 kojom se označava da datoteka već postoji.

Prikazane su i operacije vezane za brisanje datoteka i direktorija. Funkcijom `removedirs(putanja)` rekurzivno se brišu svi prazni direktoriji navedeni u putanji, funkcijom `rmdir(naziv direktorija)` briše se definirani direktorij a funkcijom `remove(ime datoteke)` briše se određena datoteka. Funkcija `removedirs` brisat će sve direktorije navedene u putanji pod uvjetom da ne sadrže datoteke ili direktorije. Javit će grešku ako posljednji direktorij naveden u putanji ne postoji, nije prazan ili nije direktorij. Funkcija `rmdir` dojavit će greške za iste slučaje a funkcija `remove` javit će greške ako datoteka ne postoji ili nije datoteka.

U sljedećem primjeru korištena je već spomenuta funkcija `access` kako bi se provjerilo postojanje datoteka ili direktorija i izbjeglo pojavljivanje navedenih greška.

```
import os, stat

def provjera_dozvola(putanja):

    postoji = (os.access(putanja, os.F_OK))
    citanje = (os.access(putanja, os.R_OK))
    pisanje = (os.access(putanja, os.W_OK))
    izvodenje = (os.access(putanja, os.X_OK))
    return(postoji, citanje, pisanje, izvodenje)


aps_putanja = os.getcwd()
print("Trenutno se nalazite u direktoriju %s s putanjom %s"\
     %(os.path.basename(aps_putanja), aps_putanja))
print("Ispis datoteka i direktorija u direktoriju %s" \
    %(os.path.basename(aps_putanja)))
print(os.listdir(path=aps_putanja))
nova_putanja = input("Upisite apsolutnu putanju direktorija \
    ili vise njih koje zelite stvoriti:")
os.makedirs(nova_putanja)
os.chdir(nova_putanja)

print("Promjena radnog direktorija u %s" %(os.getcwd()))
print("Relativna putanja od trenutnog radnog direktorija %s do %s je %s" \
    %(os.getcwd(), aps_putanja, os.path.relpath(aps_pu-
tanja, start=os.curdir)))
with open("test.txt",'w') as test:
   test.write("prvi red\ndrugi red\ntreci red")
print("\nStvorena je datoteka test.txt")
putanja_datoteke = os.path.abspath('test.txt')
print("Velicina datoteke je: %s bajta" %(os.stat(putanja_datoteke).st_size))
velicina = input("Skratiti velicinu datoteke na:")
os.truncate(putanja_datoteke,int(velicina))

print("Vrijeme posljednjeg pristupa datoteci i vrijeme posljednje promjene: \
    %s / %s" %(os.stat(putanja_datoteke).st_atime, os.stat(putanja_da-
toteke).st_mtime))
pristup = input("Upisati zeljeno vrijeme posljednjeg pristupa:")
promjena = input("i posljednje promjene:")
ntorka = (int(pristup),int(promjena))
os.utime(putanja_datoteke,ntorka)
print("posljednji pristup i posljednja promjena: %s / %s"\
     %(os.stat(putanja_datoteke).st_atime, os.stat(putanja_da-
toteke).st_mtime))

postoji1, citanje1, pisanje1, izvodenje1 = provjera_dozvola(putanja_datoteke)
print("Promjena dozvola datoteke")
os.chmod(putanja_datoteke, stat.S_IRUSR)
postoji2, citanje2, pisanje2, izvodenje2 = provjera_dozvola(putanja_datoteke)
```

```
print("Dozvole datoteke prije i poslije izmjene:\
\npostojanje: %s / %s \ncitanje: \
    %s / %s \npisanje: %s / %s \nizvodenje: %s / %s \n" \
        %(postoji1,postoji2, citanje1,citanje2, pisanje1, pisanje2, iz-
vodenje1, izvodenje2))

putanja_poveznice = aps_putanja + '/test(poveznica).txt'
putanja_cvrste_poveznice = aps_putanja + '/test(cvrsta_poveznica).txt'
os.symlink(putanja_datoteke, putanja_poveznice)
os.link(putanja_datoteke,putanja_cvrste_poveznice)
os.chdir(aps_putanja)


if os.access('Datoteka1.txt', os.F_OK):
os.rename('Datoteka1.txt','DatotekaA.txt')
else:
print("Nije moguće preimenovati datoteku jer ne postoji")
if os.access(aps_putanja+'Direktorij1/Direktorij11', os.F_OK):
os.renames('Direktorij1/Direktorij11','DirektorijA/DirektorijB')
else:
print("Nije moguće preimenovati direktorije jer ne postoje")

print("\nIspis direktorija %s i svih njegovih poddirektorija i
datoteka" %(os.path.basename(aps_putanja)))
for putanja,direktorij,datoteke in os.walk(aps_putanja,topdown=True):
print("putanja:", putanja)
print("poddirektorij",direktorij)
print("datoteke",datoteke)

if os.access(aps_putanja+'DirektorijA/DirektorijB', os.F_OK):
os.removedirs('DirektorijA/DirektorijB')
else:
print("Nije moguće izbrisati direktorije jer ne postoje")

if os.access('Direktorij2', os.F_OK):
os.rmdir('Direktorij2')
else:
print("Nije moguće izbrisati direktorij jer ne postoji")
if os.access('DatotekaA.txt', os.F_OK):
```

Slika 6. Primjer rada s funkcijama za upravljanje datotekama i direktorijima

Slijedi izlaz pokrenutog programa:

```
[marko@localhost zavrsni_python]$ python datoteke.py

Trenutno se nalazite u direktoriju zavrsni_python s putanjom
/home/marko/zavrsni_python

Ispis datoteka i direktorija u direktoriju zavrsni_python

['Datoteka1.txt', 'Direktorij2', 'proba.py', 'datoteke.py', 'Direktorij1',
'daemon.py', 'kill.py', 'procesi.py', 'stvaranje_procesa.py']

Upisite apsolutnu putanju direktorija ili vise njih koje zelite
stvoriti:/home/marko/Dokumenti/zavrsni/primjer

Promjena radnog direktorija u /home/marko/Dokumenti/zavrsni/primjer

Relativna putanja od trenutnog radnog direktorija
/home/marko/Dokumenti/zavrsni/primjer do /home/marko/zavrsni_python
je ../../../zavrsni_python



Stvorena je datoteka test.txt

Velicina datoteke je: 28 bajta

Skratiti velicinu datoteke na:14

Vrijeme posljednjeg pristupa datoteci i vrijeme posljednje promjene:
1629881163.8237271 / 1629881165.95378

Upisati zeljeno vrijeme posljednjeg pristupa:150

i posljednje promijene:2000

posljednji pristup i posljednja promjena: 150.0 / 2000.0

Promjena dozvola datoteke

Dozvole datoteke prije i poslije izmjene:

postojanje: True / True

citanje: True / True

pisanje: True / False

izvodenje: False / False

Nije moguće preimenovati datoteku jer ne postoji

Nije moguće preimenovati direktorije jer ne postoje

Ispis direktorija zavrsni_python i svih njegovih poddirektorija i datoteka

putanja: /home/marko/zavrsni_python

poddirektorij ['DirektorijA', 'Direktorij2']
```

```
datoteke ['test(cvrsta_poveznica).txt', 'proba.py', 'DatotekaA.txt',
'datoteke.py', 'daemon.py', 'kill.py', 'test(poveznica).txt', 'procesi.py',
'stvaranje_procesa.py']

putanja: /home/marko/zavrsni_python/DirektorijA

poddirektorij ['DirektorijB']

datoteke []

putanja: /home/marko/zavrsni_python/DirektorijA/DirektorijB

poddirektorij []

datoteke []

putanja: /home/marko/zavrsni_python/Direktorij2

poddirektorij []

datoteke []

Nije moguće izbrisati direktorije jer ne postoje

Nije moguće izbrisati direktorij jer ne postoji

Nije moguće izbrisati datoteku jer ne postoji
```

5. I/O tokovi

Operacijski sustav je strukturiran na način da svaki proces u izvođenju ima pristup trima posebnim datotekama: standardnom ulazu (prema zadanim postavkama povezan s tipkovnicom), standardnom izlazu (prema zadanim postavkama povezan sa zaslonom) i standardnom izlazu za greške (kao i standardni izlaz) (Kerrisk, 2010). U prethodnom primjeru prikazan je program koji za vrijeme izvođenja, od korisnika zahtjeva određene podatke (veličinu datoteke, vremenske oznake pristupa i promjene datoteke) te rezultat određenih funkcija ispisuje na zaslon. Spomenutim operacijama program je upravljao ulaznim i izlaznim tokovima.

U programskom jeziku Python, funkcija `input` služi za upravljanje ulaznim tokom podataka iz standardnog ulaza, a funkcija `print` se koristi za usmjeravanje izlaznog toka na standardni izlaz. Svaka datoteka s kojom proces komunicira tijekom izvođenja (čita, piše i slično) može se definirati kao ulazni ili izlazni tok podataka (Python Software Foundation, Built-in Functions, 2021). Prilikom otvaranja bilo koje datoteke, operacijski sustav joj pridružuje cijeli broj, opisnik datoteke (*file descriptor*), kojim proces može rukovati datotekom. Standardnom ulazu pridružen je opisnik 0, standardnom izlazu opisnik 1, a standardnom izlazu za greške opisnik 2. Svakoj sljedećoj otvorenoj datoteci pridružuje se sljedeći slobodan cijeli broj (Kerrisk, 2010).

Iako više procesa ima otvorenu istu datoteku, svaki proces ima različiti opisnik vezan uz nju. Ovisno o postavkama, proces dijete može naslijediti određene opisnike svog roditelja. Jednom kad proces prestane koristiti datoteku i zatvori ju, sustav može njezin opisnik pridružiti drugoj datoteci (Tanenbaum, 2015).

Ulazni i izlazni tokovi važni su i za komunikaciju dvaju procesa. Procesi mogu međusobno komunicirati koristeći cijevi (*pipes*) (Silberschatz, Galvin, & Gagne, 2008). Cijevi se koriste za pretvorbu izlaznog toka jednog procesa u ulazni tok drugog. Identificirane su pomoću dva opisnika, jednog za ulazni tok i jednog za izlazni. Procesi postupaju prema cijevima isto kao i prema bilo kojem drugom ulaznom ili izlaznom toku i mogu razlikovati cijevi od datoteka samo slanjem sistemskog poziva `FSTAT` (Tanenbaum, 2015).

5.1 Funkcije za upravljanje I/O tokovima pomoću opisnika datoteke

Opisi funkcionalnosti i sintakse programskog jezika Python u ovome potpoglavlju rada izrađeni su na temelju sljedećih izvora: (Miletić, 2021), (Hellman, 2011), (Python Software Foundation, Built-in Functions, 2021) i (Python Software Foundation, os — Miscellaneous operating system interfaces, 2021).

Pomoću modula os moguće je stvarati i upravljati ulazno-izlaznim tokovima procesa koristeći opisnike datoteka. Za mnoge funkcije prikazane u prethodnom primjeru postoje inačice s istom namjenom koje kao argument, umjesto putanje datoteke, primaju opisnik datoteke. U sljedećem primjeru (Slika 7) najprije je korištena funkcija `os.open(putanja, način korištenja)` kojom se stvara datoteka „`ulazno_izlazna_datoteka.txt`". Funkcija `os.open` kao argumente prima putanju datoteke (apsolutnu ili relativnu) i način korištenja datoteke. Za definiranje načina korištenja mogu se koristiti i kombinirati konstante modula `os`. U primjeru su korištene konstante `os.O_CREAT` i `os.O_RDWR` kojima se definira stvaranje datoteke te čitanje i pisanje u datoteku. Funkcija `os.open` vraća vrijednost opisnika (koji se u primjeru sprema u varijablu „`fd`" i ona se ispisuje na zaslon).

Za dupliciranje opisnika, tj. stvaranje novog opisnika za istu datoteku mogu se koristiti funkcije `os.dup(fd)` ili `os.dup2(fd, fd2, inheritable=True)`. Funkcija `dup` vraća novi opisnik (koji se ne može naslijediti) vezan uz datoteku označenu prosljeđenim opisnikom. Funkcija `dup2` omogućuje određivanje vrijednosti novog opisnika (`fd2`) i svojstva nasljeđivanja (kao što je prikazano u primjeru).

Za provjeru povezanosti ulaznog ili izlaznog toka s terminalom koristi se funkcija `os.isatty(fd)` kojoj se prosljeđuje opisnik (`fd`). Ako je tok povezan s terminalom, moguće je provjeriti kodiranje povezanog uređaja funkcijom `os.device_encoding(fd)`.

Modul os sadrži i funkcije kojima se dohvaćaju podaci o terminalu povezanim s datotekom. Funkcija `os.ttyname(fd)` vraća ime uređaja vezanog uz terminal, dok funkcijom `os.get_terminal_size(fd)` moguće je saznati broj stupaca i redaka terminala. Za navedene funkcije u primjeru su korišteni opisnici za standardni ulaz i izlaz jer ostale otvorene datoteke nisu povezane s terminalom.

Funkcija `os.fstat(fd)` ima istu namjenu (dohvaćanje podataka o datoteci) kao prethodno korištena funkcija `os.stat(putanja)` s razlikom da za identifikaciju datoteke, `fstat` koristi opisnik datoteke.

Za dohvaćanje podataka o datotečnom sustavu datoteke mogu se koristiti dvije podjednake funkcije, `os.fstatvfs(fd)` i `os.statvfs(putanja)`, koje se razlikuju jedino u načinu identifikacije datoteke (`statvfs` koristi putanju datoteke dok `fstatvfs` koristi opisnik datoteke). U primjeru, podaci dobiveni funkcijama `fstat` i `fstatvfs` su korišteni kako bi se prikazalo pisanje u datoteku „`ulazno_izlazna_datoteka.txt`" pomoću funkcija `os.write(fd,niz bajtova)` i `os.pwrite(fd, niz bajtova, pomak)`. Dohvaćene podatke je potrebno pretvoriti u niz bajtova (funkcija `str.encode(znakovni niz)`) kako bi ih funkcije za pisanje mogle koristiti. Funkciji `os.write` prosljeđuju se opisnik datoteke i niz bajtova za pisanje. Uz navedene argumente funkcija `os.pwrite` prima i pomak kojim se definira mjesto na kojem funkcija `pwrite` započne pisanje niza bajtova u datoteku. U primjeru je funkciji `pwrite` proslijeđena veličina datoteke kao vrijednost pomaka, kako bi ona svoj zapis dodala na kraj datoteke. U slučaju da je pomak veći od veličine datoteke, i funkcija upiše niz nakon završetka datoteke, dolazi do takozvane rupe u datoteci (*file hole*) (Kerrisk, 2010).

Otvaranje datoteke moguće je i predefiniranom funkcijom `open(putanja)`. Funkciji `open` potrebno je proslijediti putanju datoteke i ona vraća objekt datoteke. Opisnik objekta, vraćenog funkcijom `open`, pohranjen je u atributu `fileno()` (u primjeru je opisnik datoteke „`test(cvrsta_poveznica).txt`" spremljen u varijablu „`fd_test`").

Nakon što korisnik za otvorenu datoteku „`test(cvrsta_poveznica).txt`" ima dozvolu samo za čitanje, potrebno je promijeniti dozvole datoteke. Dozvole se pomoću opisnika datoteke mijenjaju koristeći funkciju `fchmod(fd, dozvole)`. Nakon što je korisniku omogućeno pisanje, čitanje i izvođenje datoteke, funkcijom `os.sendfile(izlazni fd, ulazni fd, pomak, broj bajtova)` prepisuje se 10 bajtova spomenute datoteke „`test(cvrsta_poveznica).txt`" u datoteku „`ulazno_izlazna_datoteka.txt`".

Za čitanje datoteke koriste se funkcije `os.read(fd, broj bajtova)` i `os.pread(fd, broj bajtova, pomak)`. Funkcija `os.read` prima opisnik datoteke i broj bajtova te vraća niz bajtova iz datoteke. Funkcija `os.pread` nudi i mogućnost određivanja pomaka, pozicije od koje će se krenuti čitati niz bajtova. Obje funkcije za čitanje kao rezultat izvođenja

vraćaju nizove bajtova koji se mogu prevesti u znakovni niz pomoću funkcije `str.decode()`.

Za zatvaranje datoteka otvorenih funkcijom `os.open` preporučeno je korištenje funkcije `os.close(fd)`, dok se za zatvaranje niza datoteka može koristiti funkcija `os.closerange(najmanji fd, najveći fd)` kojem se prosljeđuju najmanji i najveći opisnici niza datoteka koje se želi zatvoriti. Datoteke otvorene predefiniranom funkcijom `open` mogu se zatvoriti njezinom metodom `close`.

```
import os, stat

fd = os.open('/home/marko/zavrsni_python/ulazno_izlazna_datoteka.txt',\
    os.O_CREAT|os.O_RDWR)
print("Stvorena je datoteka s file descriptorom: ",fd)
fd2 = os.dup2(fd, 4, inheritable=True)

print("Kodiranje standardnog ulaza je ",os.device_encoding(0))
print("Standardni izlaz povezan s terminalom: ",os.isatty(1))
if os.isatty(1):
    print("naziv terminala: ",os.ttyname(1))
    print("terminal ima %s stupaca i %s redaka" \
        %(os.get_terminal_size(1).columns, os.get_terminal_size(1).lines))

datoteka_info = os.fstat(fd2)
datoteka_info_str = 'veličina datoteke ' + str(datoteka_info.st_size)\
     + '\nvrijeme stvaranja ' + str(datoteka_info.st_ctime)
datoteka_kodirano = str.encode(datoteka_info_str)
sustav_info = os.fstatvfs(fd2)
sustav_kodirano = str.encode('\nveličina blokova ' + str(sustav_info.f_bsize)\
     + '\nbroj datoteka ' + str(sustav_info.f_files))
os.write(fd2, datoteka_kodirano)
os.pwrite(fd2, sustav_kodirano,datoteka_info.st_size)

test = open("test(cvrsta_poveznica).txt")
fd_test = test.fileno()
os.fchmod(fd_test,stat.S_IRWXU)
print("prepisano je %s bajtova" %(os.sendfile(fd2, fd_test, 0, 10)))
kodiran2 = os.pread(fd_test, 10, 5)
print("Ispis 5 bajtova datoteke test(cvrsta_poveznica).txt ",kodiran2.de-
code())
print("Ispis 13 bajtova datoteke I/O.txt:\n", (os.read(fd2,13)).decode())
os.closerange(fd,fd2)
test.close()
```

Slika 7. Primjer rada s funkcijama za upravljanje I/O tokovima pomoću opisnika datoteke

Slijedi izlaz pokrenutog programa:

```
[marko@localhost zavrsni_python]$ python i_o.py
Stvorena je datoteka s file descriptorom:  3
Kodiranje standardnog ulaza je  UTF-8
Standardni izlaz povezan s terminalom:  True
naziv terminala:  /dev/pts/1
terminal ima 229 stupaca i 14 redaka
prepisano je 10 bajtova
Ispis 5 bajtova datoteke test(cvrsta_poveznica).txt  red
drugi
Ispis 13 bajtova datoteke I/O.txt:
  blokova 4096
```

5.2 Cijevi

Opisi funkcionalnosti i sintakse programskog jezika Python u ovome potpoglavlju rada izrađeni su na temelju sljedećih izvora: (Hellman, 2011) i (Python Software Foundation, os — Miscellaneous operating system interfaces, 2021).

Kao što je već ranije spomenuto, cijevi omogućuju komunikaciju između procesa. Modul os omogućava kreiranje cijevi između dva procesa ili između procesa i naredbenog retka (*command line*).

Idući primjer (Slika 8) prikazuje kreiranje cijevi funkcijom `os.pipe()`. Funkcija `os.pipe()` vraća dva opisnika vezana uz cijev. Prvi vraćeni opisnik namijenjen je ulaznom toku i sprema se u varijablu „`fd_r`" a drugi je namijenjen izlaznom toku i sprema se u varijablu „`fd_w`".

Za prikaz komunikacije procesa, stvara se proces dijete funkcijom `os.fork()`. Proces dijete neće koristiti ulazni tok, pa ga zatvara funkcijom `os.close`. Zadatak procesa djeteta je izvođenje naredbe „`uname -o`" u naredbenom retku te prosljeđivanje rezultata procesu roditelju. Dijete uspostavlja cijev prema naredbenom retku pomoću funkcije `os.popen(naredba,način)`. Funkciji `popen` prosljeđuje naredbu koju želi izvesti u

naredbenom retku (`uname -o`) i način upotrebe vraćenog objekta (čitanje). Proces dijete sprema vraćeni objekt u varijablu „`naredba_rez`" pomoću metode `read()` i zatvara cijev prema naredbenom retku metodom `close()`. Nakon čitanja, proces pretvara objekt u niz bajtova i funkcijom `os.write` upisuje ga u izlazni tok cijevi. Na kraju, proces dijete zatvara izlazni tok cijevi i završava izvođenje. Proces roditelj zatvara izlazni tok podataka i čeka završetak izvođenja procesa djeteta. Nakon toga, funkcijom `os.fdopen(fd)` otvara izlazni tok stvorene cijevi i metodom `read()` sprema vraćeni objekt u varijablu „`rezultat`". Nakon ispisa izlaznog toka cijevi spremljenog u spomenutoj varijabli, proces završava izvođenje.

```
import os

fd_r, fd_w  = os.pipe()
print("Otvorena cijev s file descriptorom %s za citanje i %s za pisanje" \
    %(fd_r,fd_w))
pid = os.fork()

if pid > 0:
    os.close(fd_w)
    os.wait()
    r = os.fdopen(fd_r)
    rezultat = r.read()
    print("Rezultat naredbe uname -o pokrenute funkcijom popen \
        u procesu djetetu: ", rezultat)
    os.close(fd_r)


else:
    os.close(fd_r)
    naredba=os.popen('uname -o', mode='r')
    naredba_rez = naredba.read()
    naredba.close()
    rezultat_kod=str.encode(naredba_rez)
    os.write(fd_w, rezultat_kod)
    os.close(fd_w)
    os._exit(0)
```

Slika 8. Primjer kreiranja cijevi

Slijedi izlaz pokrenutog programa:

```
[marko@localhost zavrsni_python]$ python pipe.py

Otvorena cijev s file descriptorom 3 za citanje i 4 za pisanje

Rezultat naredbe uname -o pokrenute funkcijom popen u procesu djetetu:  GNU/Linux
```

6. Zaključak

Operacijski sustav temelj je rada računala. Njegove glavne značajke su upravljanje resursima računala i povezivanje hardvera s aplikativnim programima. Programski jezik Python i njegov modul os nude razne mogućnosti za upravljanje značajkama operacijskog sustava. Jedna od glavnih prednosti modula os je stvaranje i upravljanje procesima operacijskog sustava. Proces je instanca programa u izvođenju i jedan od osnovnih načina na koji operacijski sustav ispunjava svoje zadaće. Uz stvaranje i upravljanje procesima, modul os nudi i niz funkcija kojima se mogu mijenjati značajke i podaci trenutnog procesa i korisnika.

Modul os sadrži i razne funkcije vezane uz datotečni sustav operacijskog sustava. Datotečni sustav je dio operacijskog sustava koji je namijenjen pohrani i upravljanju podacima. Funkcijama modula os omogućeno je izvršavanje raznih operacija nad datotekama i direktorijima te manipulacija putanjama datoteka. Ulazno-izlazni tokovi su također jedna važna značajka operacijskog sustava. Oni omogućuju procesima dohvat i pohranu podataka te komunikaciju s ostalim procesima. Modul os ima implementirane razne funkcije kojima upravlja tokovima podataka pomoću njihovih opisnika datoteka.

Na temelju svega navedenog, može se uvidjeti zašto je modul os jedan od ključnih modula programskog jezika Python za korištenje funkcionalnosti operacijskog sustava.

7. Literatura

Both, D. (7. Rujan 2021). *An introduction to Linux's EXT4 filesystem*. Dohvaćeno iz opensource: https://opensource.com/article/17/5/introduction-ext4-filesystem

File security. (18. Kolovoz 2021). Dohvaćeno iz Linux documentation: https://linux.die.net/Intro-Linux/sect_03_04.html#sect_03_04_02_02

Hellman, D. (2011). *The Python Standard Library by Example.* Addison-Wesley Professional.

Kerrisk, M. (2010). *The Linux Programming Interface: A Linux and UNIX System Programming Handbook.* No Starch Press.

Kovačić, B. (2008). *Operacijski sustavi.* Rijeka.

Miletić, V. (16. Srpanj 2021). *Python: općenite usluge operacijskog sustava: osnovna sučelja*. Dohvaćeno iz https://group.miletic.net/hr/nastava/materijali/python-modul-os/

Python Software Foundation. (19. Srpanj 2021). *Built-in Functions*. Dohvaćeno iz https://docs.python.org/3/library/functions.html#open

Python Software Foundation. (5. Svibanj 2021). *os — Miscellaneous operating system interfaces*. Dohvaćeno iz https://docs.python.org/3/library/os.html#process-parameters

Python Software Foundation. (15. Kolovoz 2021). *PEP 8 -- Style Guide for Python Code*. Dohvaćeno iz https://www.python.org/dev/peps/pep-0008/

Python Software Foundation. (1. Svibanj 2021). *Python Success Stories*. Dohvaćeno iz https://www.python.org/about/success/

Python Software Foundation. (29. Travanj 2021). *What is Python? Executive Summary*. Dohvaćeno iz https://www.python.org/doc/essays/blurb/

Real, Effective and Saved UserID in Linux. (17. Srpanj 2021). Dohvaćeno iz GeeksforGeeks: https://www.geeksforgeeks.org/real-effective-and-saved-userid-in-linux/

Silberschatz, A., Galvin, P. B., & Gagne, G. (2008). *Operating system concepts (8th edition).* Hoboken: Wiley.

Slavuj, V. (2021). Rad s procesima (nastavni materijali). Rijeka.

Tanenbaum, A. S. (2015). *Modern Operating Systems (fourth edition).* Pearson.

Techopedia. (1. Svibanj 2021). *System Programming*. Dohvaćeno iz https://www.techopedia.com/definition/9616/system-programming

The Operating System. (18. Srpanj 2021). Dohvaćeno iz Debian Policy Manual v4.6.0.1: https://www.debian.org/doc/debian-policy/ch-opersys.html#uid-and-gid-classes

TIOBE Software BV. (8. Svibanj 2021). *TIOBE Index*. Dohvaćeno iz TIOBE Index: https://www.tiobe.com/tiobe-index/

8. Popis slika