Perianu Leon
ANUL 3/grupa 3
INFORMATICĂ APLICATĂ

1.a. Un proces UNIX este identificat printr-o serie de valori printre care valoarea (unică) PID [Process id]

1.b. ~~Grupul de user~~ Identificatorul de grupe se face cu funcția setuid(idNou)

1.c. Pentru a determina proprietarul unui fișier se folosește o verificare permisiunile de acces la fișier și a verifica permisiunile de transmitere a semnalelor.

2.a. Procesele sunt create cu funcția fork(). Această funcție returnează -1 în caz de eroare dar o valoare în caz de succes.

2.b. se pune un if în care se verifică valoarea returnată de fork. fork returnează id-ul procesului copil la părinte și 0 la copil.

```
if(fork()==0){
    copil
}
părinte
```

2.c. a. În cazul în care nu avem suficientă memorie, atunci fork returnează -1.

3.a. fork - copiază programul chemător-i un spațiu de memorie care conține o copie a resurselelor de până acum. Programul reporă execuția verificând valoarea returnată de fork().

exec - Șterge contextul vechiului program, execuția instrucțiunilor de după exec(ARG, NULL) nu vor mai fi executate.

wait - oprește programele să ajungă în acest punct înainte procesul. Acesta va culege informații de către resursele după execuția programelor

Pthread_create - Crează un fir de execuție care are un spațiu de memorie comun cu procesul creator.

Pthread_join - suspendă activitatea programului până când firul așteptat își încheie execuția (similar cu wait)

3.b - fork() oferă un spațiu de memorie individual fiecărui proces, Pthread_create folosește un spațiu de memorie comun.

- reporirea firelor de execuție în fork se face prin verificarea valorii de return a lui fork() în timp ce la Pthread_create se dă adresa de memorie a funcției executate

- Fork consumă mai multă memorie decât un Pthread.

3.c wait(NULL) va aștepta terminarea tuturor proceselor

3.d. detașarea unui thread îl face să nu mai fie joinable. Acest lucru poate face ca un thread să rămână activ chiar și după terminarea programului.

4.a. Pentru redirectare folosim dup(int Fd). (File de scriptor)
- redirectarea ieșirii presupune modificarea înțelesului dispozitivului stdout
- operații de redirectare:
  1. deschiderea unui fișier compatibil cu fișierul redirectat (scriere)
  2. închiderea dispozitivului redirectat
  3. duplicarea descriptorului identificat (1)
  4. închiderea descriptorului original (1)

4.b deschidem fișierul primit ca parametru.
se închide dispozitivul redirectat
se duplică identificatorul de la fișier
se închide descriptorul inițial

4.c ~~Fo~~

5.a Folosind bind(sd, adresa, addr_len) se asociază un anumit ~~port~~ adresă pentru ascultare

5.b Socketurile UNIX folosesc un sistem de fișiere în locale un protocol de rețea pentru toate comunicațiile în cadrul Kernelului

5. c pentru TCP dacă recvFrom returnează -1 atunci știm că am pierdut conexiunea cu serverul.
În cazul UDP verificarea conexiunii nu se poate face la fel ca la TCP, dar putem seta un timeout.

6. a Putem folosi comanda unbuffer pentru a opri bufferul pentru intsoll și iesill. FFlush() curăță bufferul curent.

6. b - select stabilește o așteptare pentru ca descriptorii din una sau mai multe dintre mulțimile specificate să fie capabili să suporte operația precizată.
- așteptarea poate fi nelimitată, limitată sau inexistentă
- mulțimile de descriptori se construiesc folosind macrouri apelul select
- apelul va returna numărul de descriptori pregătiți pentru operații / o când cronometrul expiră

6. c Putem folosi select pentru verificarea disponibilității fișierelor în cele 3 locuri diferite, și în cazul în care sunt date atunci le copiem unde dorim. Se poate stabili un timer între verificări.

7.a Un semnal nu transmite date
sig USR1 poate fi folosit de utilizator sig INT este folosit pentru terminarea procesului
Se face un handler în codul unui proces
{ signal(SIGUSR1, adresa funcției);
Pause() } → în caz de primire semnal funcția va fi executată.
Semnale pot fi transmise prin socketuri, pipe, signal, adresa de memorie partajată.

7.b Putem scrie într-un fișier starea procesului sau putem folosi signal pentru a seta starea

7.c Într-o aplicație multifir definim handle-ul pentru semnal în procesul copil

signal(SIGUSR1, Funct1)
if(Fork() == 0){
signal(SIGUSR1, Funct2)} → în acest mod cele două procese pot răspunde în mod diferit la semnalul SIGUSR1