User Schedule (Round Robin)

Fronescu Martin-Cristian, Nazare Elena-Denisa

Universitatea din Bucuresti
Departamentul de Informatica
Sisteme de operare
2023-2024

Cuprins

- Descrierea problemei
- Specificatia solutiei
- Design solutie
- Implementare
- Experimente
- Concluzii

Descrierea problemei

- Problema: Planificarea si executarea mai multor procese, a mai multor utilizatori
- Este o problema de tip planificare de procese
- scopul este eficientizarea rularii lor pe CPU, distribuind resursele astfel incat sa nu trebuiasca un utilizator sa astepte sa se execute toate procesele unui alt utilizator
- Folosim un algoritm Round Robin, mai exact Weighted Round Robin

Specificatia solutiei

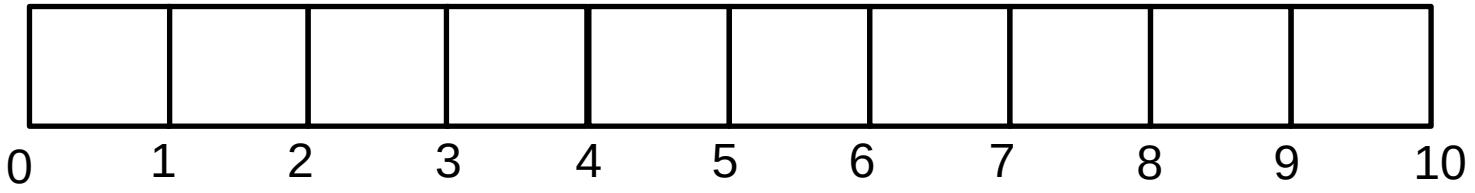
- Mai exact, avand o lista de utilizatori si ponderile lor, cat si coada de procese pe care vor sa le ruleze, trebuie sa implementam un algoritm care sa le planifice executia astfel incat sa fie relativ balansat timpul de executie intre utilizatori (bazat totusi pe ponderi)
- Tinta programului este ca toate procesele utilizatorilor sa se execute.
- Programul este implementat in C si creeaza procese copii pentru a executa procesele utilizatorilor.
- Este o problema specifica pentru kernel, neavand posibilitatea programarii kernelului, am realizat o simulare.
- Programul ruleaza pe Linux. Memoria necesara depinde de numarul de useri si numarul de procese ale acestora. Memoria utilizata este de aceasi complexitate ca datele de intrare. Timpul de executie depinde de timpul de executie al proceselor ce trebuie rulate. Nu s-au utilizat biblioteci software externe, doar cele standard limbajului C

Specificatia solutiei

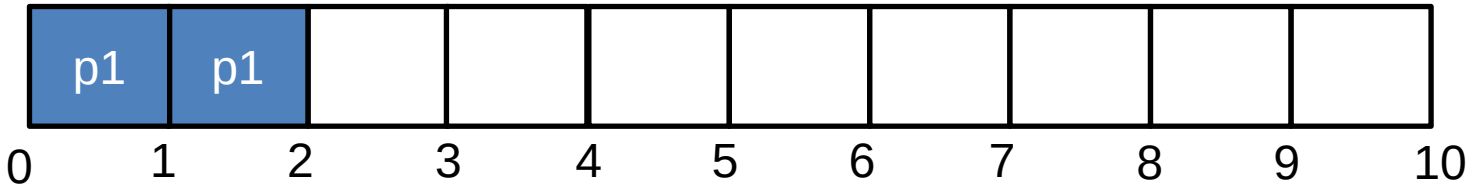
- Procesele ce vor fi rulate trebuie sa se afle in acelasi director cu executabilul, datele de intrare fiind furnizate prin fisierul input.txt din director.
- Limitari:
  - Programul presupune un numar maxim de utilizatori, precum si de procese, dar acest numar se poate modifica in cod.
  - Cuanta alocata unui proces intr-un ciclu este un numar intreg de secunde, din cauza limitarii functiei alarm.

Design

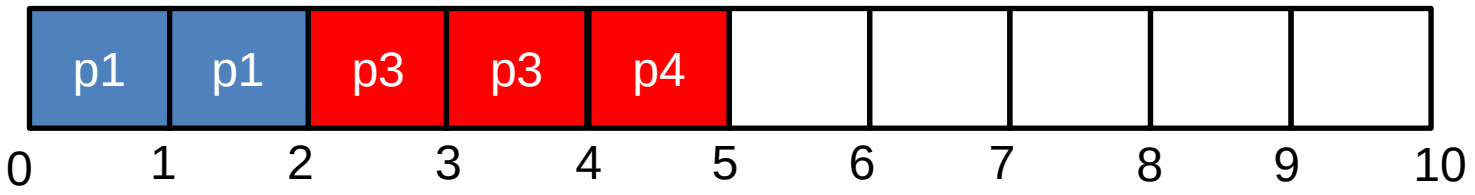
- Programul utilizeaza algoritmul Weighted Round Robin, ponderile fiind date de useri. Astfel in functie de aceste ponderi, fiecare user are o cuanta de timp de executie per ciclu. Algoritmul executa procesele primului user cat timp acesta are alocat, apoi aceste procese sunt intrerupte si se continua cu urmatorul utilizator in acelasi mod. Cand s-a parcurs intreaga lista de utilizatori, algoritmul se intoarce la primul utilizator (daca acesta mai are procese de executat), si continua executia proceselor lui, cu aceasi limita de timp ca si la primul ciclu. La fel si cu restul utilizatorilor. Acest proces se repeta pana cand toti utilizatorii au toate procesele rulate.
- Am ales sa facem Round Robin pe utilizatori, insa puteam sa facem si pe procesele fiecarui user.



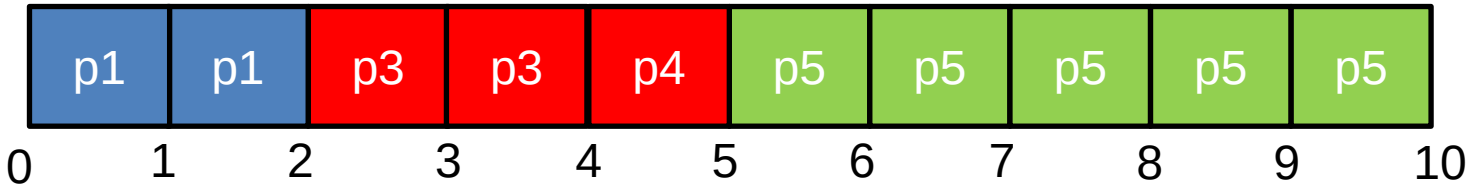
- USR1: 20% (2s)
  - p1 – 4s
  - p2 – 2s
- USR2: 30% (3s)
  - p3 – 2s
  - p4 – 2s
- USR3: 50% (5s)
  - p5 – 6s
- Timp total 10s (per ciclu)



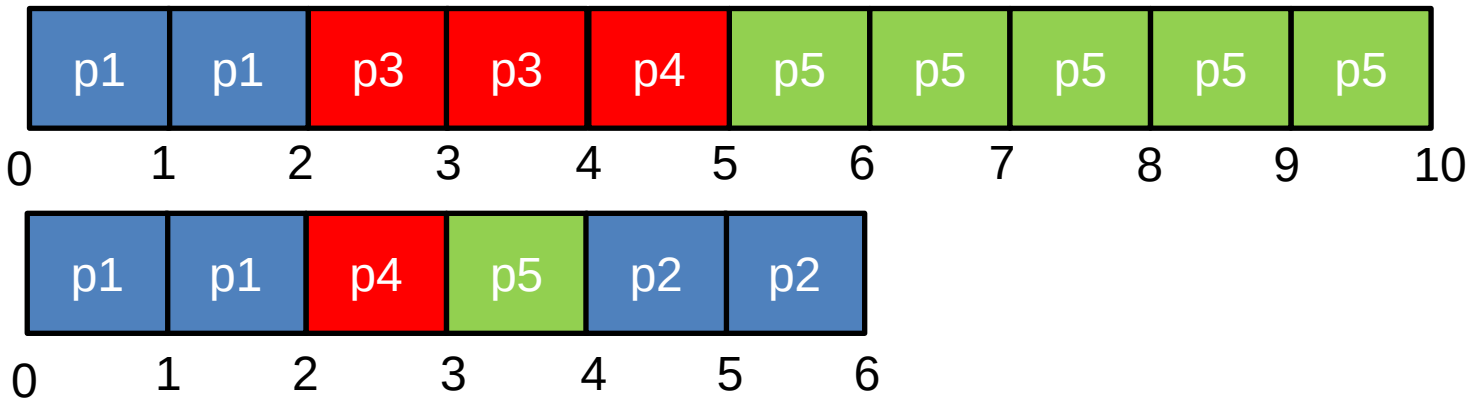
- USR1: 20% (2s)
  - p1 – 4s
  - p2 – 2s
- USR2: 30% (3s)
  - p3 – 2s
  - p4 – 2s
- USR3: 50% (5s)
  - p5 – 6s
- Timp total 10s (per ciclu)



- USR1: 20% (2s)
  - p1 – 4s
  - p2 – 2s
- USR2: 30% (3s)
  - p3 – 2s
  - p4 – 2s
- USR3: 50% (5s)
  - p5 – 6s
- Timp total 10s (per ciclu)



- USR1: 20% (2s)
  - p1 – 4s
  - p2 – 2s
- USR2: 30% (3s)
  - p3 – 2s
  - p4 – 2s
- USR3: 50% (5s)
  - p5 – 6s
- Timp total 10s (per ciclu)



- USR1: 20% (2s)
  - p1 – 4s
  - p2 – 2s
- USR2: 30% (3s)
  - p3 – 2s
  - p4 – 2s
- USR3: 50% (5s)
  - p5 – 6s
- Timp total 10s (per ciclu)

Implementare

- Un vector de structura de tip user pentru a stoca informatiile necesare
- Un vector care retine pid-ul la care a fost intrerupta executia pentru fiecare user.
- Un vector care contine indexul procesului la care fiecare user a ramas
- Fiecare user are un vector de stari ale proceselor lui (0 – nu a fost executat, 1 executie neterminata, 2 executie terminata)
- Citirea se face din fisierul “input.txt” si memoria e alocata dynamic in mare parte, cu malloc.
- Procesele au fost executate cu execv, mai intai realizandu-se un proces copil prin fork.
- Am folosit alarm pentru a permite proceselor unui user sa ruleze doar cat timp are acesta alocat. Mai exact, signal handler-ul pentru SIGALRM trimite SIGSTOP procesului care rula in momentul acela. Cand ne intoarcem la un user, verificam statusul procesului la care a ramas, pentru a sti daca il executam normal, sau ii dam SIGCONT.

```c
int current_process_index[MAX_USERS];
volatile pid_t current_process_pid[MAX_USERS];

struct User {
    char* name;
    int weight;
    int num_processes;
    char** processes;
    int* proc_statuses; // status of each process
    // 0 - the process hasn't started,
    // 1 - the process needs to be continued,
    // 2 - the process has finished its execution
};

volatile struct User users[MAX_USERS];
```

```c
void execute_process(char* user, char* process) {
    char cwd[PATH_MAX], path[PATH_MAX];

    if (getcwd( buf: cwd,  size: sizeof(cwd)) == NULL) { // gets the current directory
        perror( s: "getcwd");
        _exit( status: 1);
    }
    snprintf( s: path,  maxlen: sizeof(path),  format: "%s/%s", cwd, process);

    printf( format: "Executing %s: %s\n", user, process);
    fflush( stream: stdout);

    char* program_args[] = { [0]: path,  [1]: NULL};

    execv( path: program_args[0],  argv: program_args); // executes the process

    perror( s: "execv");
    _exit( status: 1);
}
```

```c
94
95      void alarm_handler(int signum) {
96
97          change_user = true;
98          kill( pid: child_pid,  sig: SIGSTOP); // suspends the current process execution because the user's time is up
99
100         if (users[global_current_user].proc_statuses[global_current_process[global_current_user]] == 0) {
101             users[global_current_user].proc_statuses[global_current_process[global_current_user]] = 1;
102             // if the process wasn't executed before, and has been suspended,
103             // we mark it as 1 because it needs to be continued.
104         }
105     }
```

Implementare

Tot ce am mentionat mai sus, este intr-un for loop unde se executa procesele pentru fiecare utilizator, iar tot for-ul este intr-un while pentru a repeta continuu ciclul de utilizatori, pana cand starea tuturor proceselor este 2.

```
115     bool is_finished_exec(){
116         // returns true when all users have finished executing all their processes
117         for(int i=0; i< NUM_USERS;i++)
118             for(int j=0;j<users[i].num_processes;j++)
119                 if(users[i].proc_statuses[j]!=2)
120                     return false;
121         return true;
122     }
123
124     void user_weighted_round_robin() {
125         while (!is_finished_exec()) {
126             for (int i = 0; i < NUM_USERS; ++i , change_user = false) {
```

Implementare

- Problema intampinata:
  - Initial voiam sa setam starea unui proces in 1 in momentul cand acesta e intrerupt, prin signal handle pentru SIGSTOP. Problema este ca SIGSTOP si SIGCONT nu pot fi handle-uite.

- Solutie:
  - Am folosit waitpid cu WUNTRACED pentru a astepta si SIGSTOP, nu doar terminarea normala a executiei copilului, iar valoarea returnata in status a fost verificata cu WIFSTOPPED si WIFEXITED pentru a afla daca procesul s-a terminat sau nu (starea procesului 1 sau 2)

- Limitarile implementarii furnizate: user-ul poate fi limitat doar la un numar intreg de secunda din cauza utilizarii alarm.

```c
if (pid == 0) {
    // child process
    execute_process( user: current_user->name,
                     process: current_user->processes[global_current_process[global_current_user]]);
    _exit( status: 0);
} else if (pid > 0) {
    // parent process
    int status;
    pid_t stopped_pid = waitpid(pid,  stat_loc: &status,  options: WUNTRACED);
    // WUNTRACED to wait for when the child finishes because of SIGSTOP received
    // we use status to determine whether it was stopped or just finished executing normally

    if (stopped_pid == -1) {
        perror( s: "waitpid");
        exit( status: 1);
    }


    if (WIFSTOPPED(status)) {
        current_process_pid[i] = stopped_pid;
        // if process was stopped, mark its status as 1
        users[global_current_user].proc_statuses[global_current_process[global_current_user]] = 1;
    } else if (WIFEXITED(status)) {
        // if process was stopped, mark its status as 2
        users[global_current_user].proc_statuses[global_current_process[global_current_user]] = 2;
    }
```

Limitari legate de implementare

- Numarul maxim de utilizatori, precum si de procese sunt hardcodate
- Cuanta alocata unui proces intr-un ciclu este un numar intreg de secunde, din cauza limitarii functiei alarm.
- Durata unui ciclu este hardcodata.

Experimente

- Arhitectura hard/soft folosita pt testarea solutiei
  - Masina virtuala Ubuntu 22.04.03 LTS: 2 core-uri, 4GB ram
  - nr de procese depinde de input
  - Memoria folosita depinde de la caz la cat deoarece memoria alocata unui user este dinamica.
- Am testat programul creand 9 procese ce ruleaza un numar diferit de secunde, fiecare proces afisand un mesaj la fiecare secunda cu numele procesului si la a cata secunda din executie este.

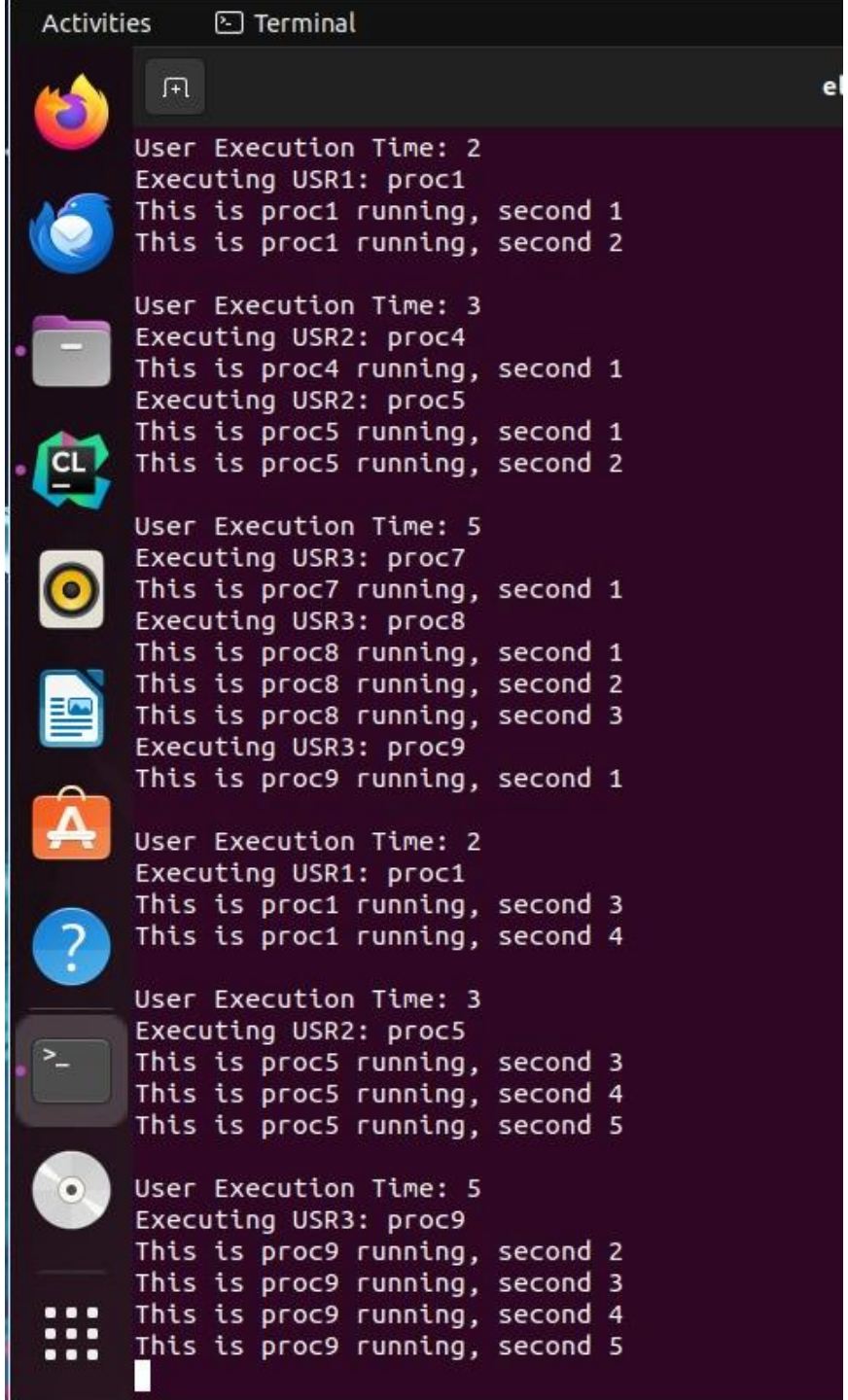
Experimente

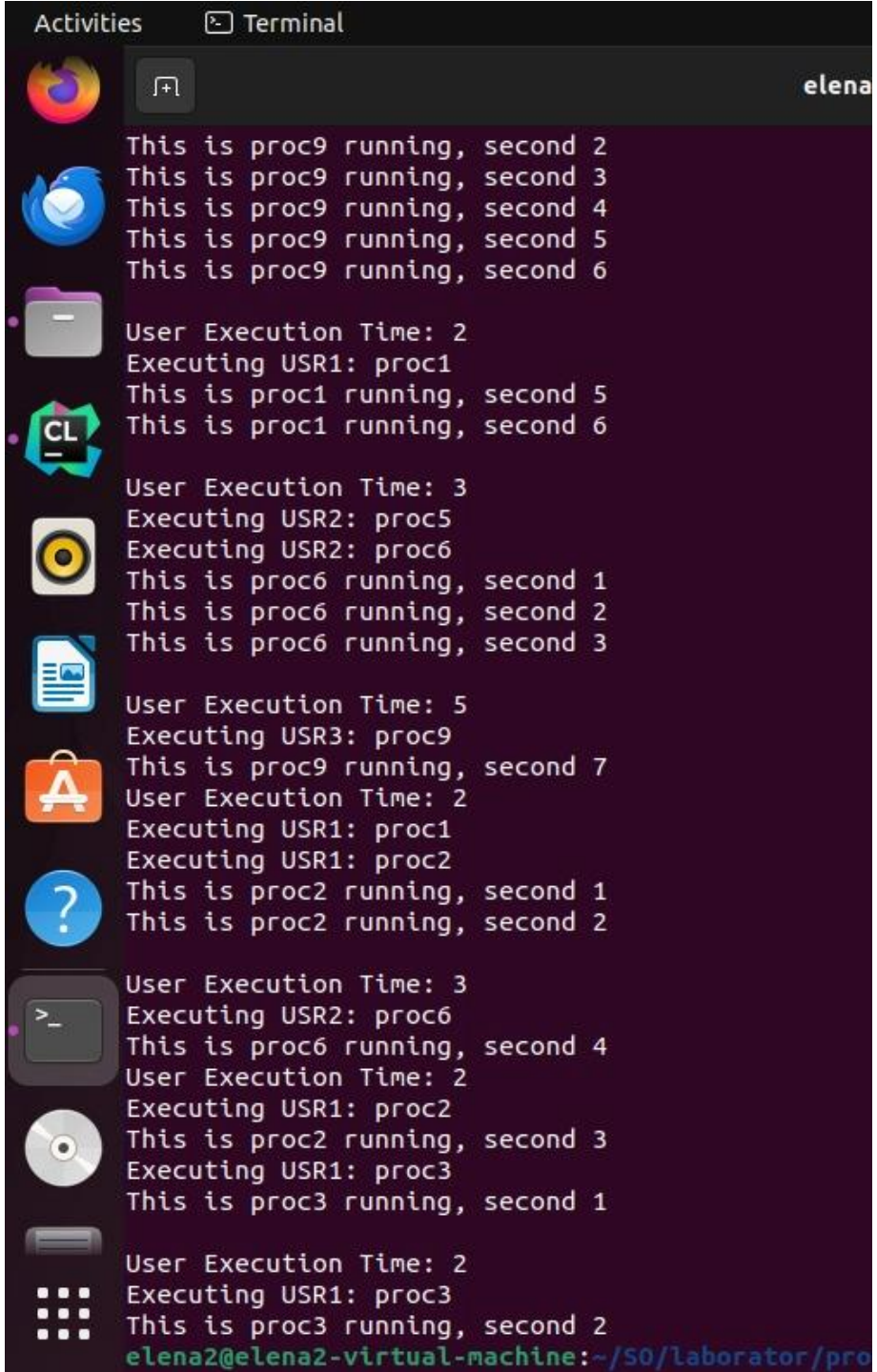
Mai exact, cele 9 procese durau:

- proc1 6s
- proc2 3s
- proc3 2s
- proc4 1s
- proc5 5s
- proc6 4s
- proc7 1s
- proc8 3s
- proc9 7s

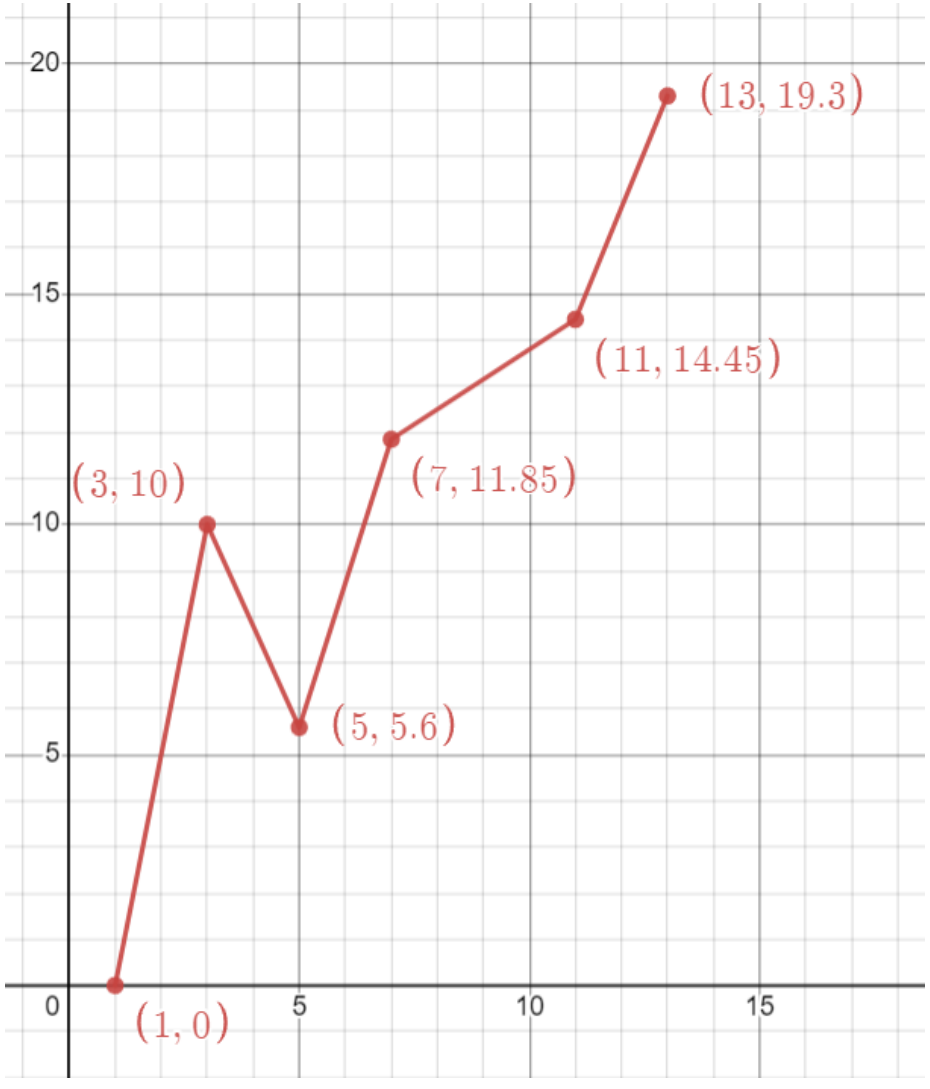
Un exemplu de input:

USR1 20 3 proc1 proc2 proc3
USR2 30 3 proc4 proc5 proc6
USR3 50 3 proc7 proc8 proc9


Activities
Terminal
User Execution Time: 2
Executing USR1: proc1
This is proc1 running, second 1
This is proc1 running, second 2
User Execution Time: 3
Executing USR2: proc4
This is proc4 running, second 1
Executing USR2: proc5
This is proc5 running, second 1
This is proc5 running, second 2
User Execution Time: 5
Executing USR3: proc7
This is proc7 running, second 1
Executing USR3: proc8
This is proc8 running, second 1
This is proc8 running, second 2
This is proc8 running, second 3
Executing USR3: proc9
This is proc9 running, second 1
User Execution Time: 2
Executing USR1: proc1
This is proc1 running, second 3
This is proc1 running, second 4
User Execution Time: 3
Executing USR2: proc5
This is proc5 running, second 3
This is proc5 running, second 4
This is proc5 running, second 5
User Execution Time: 5
Executing USR3: proc9
This is proc9 running, second 2
This is proc9 running, second 3
This is proc9 running, second 4
This is proc9 running, second 5

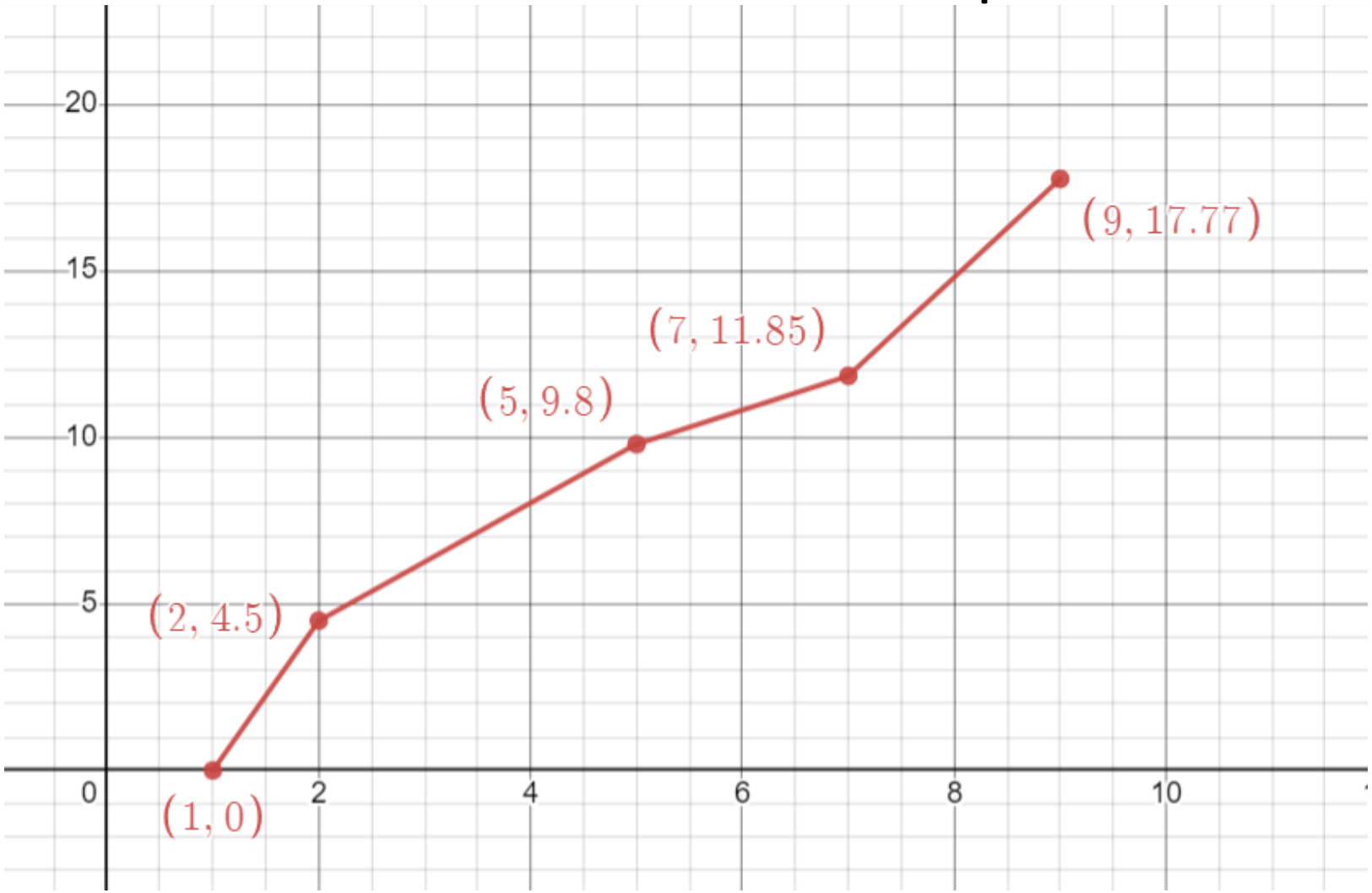

Activities
Terminal
elena
This is proc9 running, second 2
This is proc9 running, second 3
This is proc9 running, second 4
This is proc9 running, second 5
This is proc9 running, second 6
User Execution Time: 2
Executing USR1: proc1
This is proc1 running, second 5
This is proc1 running, second 6
User Execution Time: 3
Executing USR2: proc5
Executing USR2: proc6
This is proc6 running, second 1
This is proc6 running, second 2
This is proc6 running, second 3
User Execution Time: 5
Executing USR3: proc9
This is proc9 running, second 7
User Execution Time: 2
Executing USR1: proc1
Executing USR1: proc2
This is proc2 running, second 1
This is proc2 running, second 2
User Execution Time: 3
Executing USR2: proc6
This is proc6 running, second 4
User Execution Time: 2
Executing USR1: proc2
This is proc2 running, second 3
Executing USR1: proc3
This is proc3 running, second 1
User Execution Time: 2
Executing USR1: proc3
This is proc3 running, second 2
elena2@elena2-virtual-machine:~/SO/laborator/pro

Grafic timpul de asteptare mediu in raport cu nr de procese



3 useri: ponderile 30% 30% 40%

Grafic timpul de asteptare mediu in raport cu nr de procese



2 useri:
ponderile 50% 50%

Concluzii

Aspecte importante pe care le-am remarcat lucrand la acest proiect:

- Cum se folosesc SIGSTOP si SIGCONT, si faptul ca e posibil ca un proces sa fie suspendat si continuat (Ctrl S, Ctrl Q)
- Alocare dinamica in C
- Variabilele ce se modifica in handler ar trebui sa fie volatile.
- Waitpid are o variabila status prin care putem afla folosind WIFEXITED, WIFSTOPPED daca procesul a fost oprit cu SIGSTOP sau SIGINT