10/100 Bălescu Alexandru Gr 245

SISTEME DE OPERARE
EXAMEN SESIUNE DE RESTANTE
8 IUNIE 2023

INSTRUCTIUNI

- Va rog sa va scrieti numele pe fiecare foaie pe care o predati (inclusiv pe foile cu subiectele de examen pe care le veti returna impreuna cu celelalte foi)
- Examenul se tine fara documentatie pe masa si fara acces la echipamente electronice (telefon mobil, tableta, etc)
- Aveti 100 minute pentru a termina examenul. Abordati examenul cu inteligenta: daca nu stiti pe moment raspunsul la o intrebare, treceti la urmatoarea si reveniti mai tarziu; dati raspunsuri concise si evitati sa pierdeti vremea furninzand detalii irelevante sau care nu sunt solicitate.
- Primele 10 minute sunt destinate citirii subiectelor. In tot acest timp nu aveti voie sa va atingeti de ustensilele de scris. Nerespectarea acestei conditii se pedepseste cu iesirea din examen si pierderea punctajului aferent.
- Examenul se promoveaza prin obtinerea a minimum 50 de puncte din cele 100 posibile.

SUBIECTE

1. (16 pcte) Solutia problemei readers/writers (cititori/scriitori) prezentata la curs favorizeaza cititorii. Explicati in ce sens e vorba de favorizare si gasiti o solutie care favorizeaza scriitorii. Prezentati pseudocodul acestei solutii folosind semafoare.

2. (18 pcte) Sincronizarea proceselor.

   (a) (4 pcte) Care sunt conditiile necesare unei solutii corecte de partajare a resurselor?

   (b) (2 pcte) Care sunt limitarile alternantei stricte ca metoda software de sincronizare intre doua procese?

   (c) (3 pcte) Ce probleme ale strictei alternante rezolva solutia lui Peterson? Raman si probleme nerezolvate?

   (d) (3 pcte) Ce problema apare cand un proces aflat intr-un monitor apeleaza operatia **signal** asupra unei variabile conditie ? Care sunt solutiile posibile?

   (e) (6 pcte) Monitoarele in Java se implementeaza folosind cuvantul cheie **synchronized**. O metoda etichetata **synchronized**, de ex. *public synchronized void method()*, se executa in regim de excludere mutuala fata de celelalte metode sincronizate ale aceluiasi obiect. De asemenea, limbajul implementeaza metodele **wait** si **notify** care atunci cand sunt apelate din interiorul unei metode sincronizate au comportamentul apelurilor **wait** si respectiv **signal** pentru variabile conditie. Implementati in Java o clasa **semaphore** care atunci cand este instantiata are comportamentul unui semafor numarator (mai exact, implementati metodele **down** si **up** ale clasei **semaphore** care corespund comportamentului primitivelor **down** si **up** ale semafoarelor numaratoare).

3. (10 pcte) Sisteme multiprocesor.

(a) (4 pcte) Cum se clasifica arhitecturile multiprocesor din punctul de vedere al accesului la memorie? Care sunt diferentele intre aceste arhitecturi?

(b) (4 pcte) Considerati un sistem multiprocesor simetric care foloseste un protocol *write-invalidate*. De ce e TATAS (Test-And-Test-And-Set) mai potrivita pentru implementarea sectiunilor critice decat TAS (Test-And-Set) si in ce conditii?

(c) (2 pct) Ce se schimba daca se foloseste TATAS in conjunctie cu un protocol *write-update*?

4. (14 pcte) Planificarea proceselor.

(a) (4 pcte) Demonstrati ca data fiind o colectie de joburi cu timpi de rulare cunoscuti a priori, algoritmul de planificare Shortest Job First (SJF) este optimal in sensul in care planificarea obtinuta are cel mai mic timp mediu de asteptare dintre toate planificarile posibile pentru colectia de joburi data. (*Indicatie*: porniti de la formula timpului mediu de asteptare si minimizati functia respectiva).

(b) (4 pcte) Daca job-urile care trebuie planificate nu sunt disponibile simultan (nu au toate acelasi timp de sosire in sistem), mai este optimal algoritmul de planificare Shortest Job First (SJF)? Justificati raspunsul (simplul raspuns da/nu nu se puncteaza).

(c) (6 pcte) Data fiind o colectie de procese cu caracteristicile de mai jos, trasati o diagrama Gantt pentru o politica de planificare soft real-time de tip Round-Robin cu prioritati si calculati timpul mediu de asteptare pentru aceasta politica. Timpii de sosire/rulare sunt exprimati in ms, valorile mici exprima prioritati mari iar cuanta de timp a sistemului este de 1 ms. Un proces nu poate pierde procesorul decat la sfarsitul cuantei de timp alocate. Politica de planificare Round-Robin se aplica proceselor disponibile pentru planificare din aceeasi clasa de prioritate.

Proces	Timp de sosire	Timp de rulare	Prioritate
P1	0	6	2
P2	2	1	1
P3	2	2	3
P4	4	3	1
P5	0	5	2

P2 P5 P1 P5 P3

5. (10 pcte) Mecanismul de segmentare/paginare la Intel IA-32 (arhitecturi pe 32 de biti).

(a) (2 pcte) Explicati schema generala de transformare a adresei logice/virtuale in cea fizica.

(a) (4 pcte) Cum se obtine adresa liniara din adresa logica?

(b) (4 pcte) Cum se obtine adresa fizica din adresa liniara?

6. (10 pcte) Sisteme de fisiere/stocare a datelor.

(a) (2 pcte) Ce este un File Control Block (FCB)? Ce tip de informatii contine?

(b) (4 pcte) Ce este un buffer cache? Dar un page cache? Ce problema apare cand ambele cache-uri exista in sistem si care ar fi o solutie posibila?

(c) (3 pcte) Care sunt componentele principale ale timpului de acces la date pentru discuri clasice organizate pe cilindri, piste si sectoare? Detaliati raspunsul.

(d) (1 pct) Care componenta dintre cele de la punctul anterior poate fi optimizata folosind algoritmi de planificare de disc?

7. (10 pcte) Controlul proceselor.

(a) (5 pcte) Ce face urmatorul program C?

```
void handler(int);
int main(int argc, char *argv[], char *envp[])
{
        pid_t pid;
        void (*old_handler)(int);

        if((old_handler = signal(SIGCHLD, handler)) == SIG_ERR)
                perror("signal"), exit(1);
        if((pid = fork()) == -1)
                perror("fork"), exit(1);
        if(pid)
                printf("press any key: \n\n"), getchar(), printf("done\n");
        else {
                printf("%d, %d\n", getpid(), getppid());
                exit(44);
        }
        exit(0);
}
void handler(int no)
{
        pid_t pid;
        int s;
        if((pid = waitpid((pid_t)0, &s, WNOHANG)) == -1)
                        return;
        printf("%d\n", s);
}
```

(b) (5 pcte) Ce face urmatorul program C? (*Indicatie*: apelul sistem **pipe** creeaza un canal de comunicatie **unidirectional** intre doua procese; procesele pot citi din canalul de comunicatie

folosind descriptorul de fisier 0 al parametrului furnizat apelului sistem **pipe**, respectiv pot scrie in canalul de comunicatie folosind descriptorul de fisier 1 al aceluiasi parametru de apel).

```
int fd[2];
char c;
int main(int argc, char *argv[], char *envp[])
{
        pid_t pid;

        if(pipe(fd) < 0)
                perror("pipe"), exit(1);
        if((pid = vfork()) < 0)
                perror("vfork"), exit(1);
        else if(!pid) {
                read(fd[0], &c, 1);
                _exit(0);
           }
        write(fd[1], "a", 1);
        exit(0);
}
```

8. (12 pcte) Algoritmi de inlocuire a paginilor.

Se da urmatorul sir de referinte la pagini intr-un sistem cu 3 frame-uri de memorie:

3 2 1 0 3 2 4 3 2 1 0 4

(a) (2 pcte) Cate page-fault-uri inregistreaza rularea algoritmului FIFO de inlocuire a paginilor care trebuie scoase din memorie atunci cand o noua pagina trebuie incarcata si nu mai exista niciun frame liber in memorie?

(b) (5 pcte) Cum se schimba numarul de page-fault-uri daca se mareste numarul de frame-uri de memorie la 4? Cum explicati fenomenul?

(c) (5 pcte) Care este numarul de page-fault-uri daca se foloseste algoritmul optimal de inlocuire a paginilor pentru dimensiuni ale memoriei de 3 respectiv 4 frame-uri? Explicati rezultatul.