Espai grapa

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

⊂05.566ℜ19ℜ01ℜ19ℜEΞ3∈
05.566 19 01 19 EX

Enganxeu en aquest espai una etiqueta identificativa
amb el vostre codi personal
Examen

Fitxa tècnica de l'examen

- Comprova que el codi i el nom de l'assignatura corresponen a l'assignatura matriculada.
- Només has d'enganxar una etiqueta d'estudiant a l'espai corresponent d'aquest full.
- No es poden adjuntar fulls addicionals, ni realitzar l'examen en llapis o retolador gruixut.
- Temps total: **2 hores** Valor de cada pregunta: **2,5 punts**
- En cas que els estudiants puguin consultar algun material durant l'examen, quins són? **Un full mida foli/DIN-A4 amb anotacions per les dues cares** En cas de poder fer servir calculadora, de quin tipus? **PROGRAMABLE**
- Si hi ha preguntes tipus test: Descompten les respostes errònies? **NO** Quant?
- Indicacions específiques per a la realització d'aquest examen:

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

Enunciats

1. Teoria [2.5 punts]

Contesteu **justificadament** les següents preguntes:

a) Indiqueu les transicions d'estat que pot realitzar un procés que està en l'estat de *Wait*. Indiqueu també les transicions que no pot realitzar. Justifiqueu perquè no es pot produir aquestes transicions.

Un procés que està en l'estat de Wait pot passar als següents estats:

- A l'estat de End/Finalitzat si l'usuari (Ctrl + C) o un altre procés envia un senyal que finalitza aquest procés.
- A l'estat de Ready/Preparat quan es produeix l'esdeveniment pel qual estava esperant el procés (es completa la E/S, etc.).

Un procés que està en l'estat de Wait no pot passar als següents estats: No pot tornar a l'estat de nou, a causa que ja ha estat arrencat.

- No pot anar a l'estat Run/Execució directament ja que per abans ha de passar per la cua de preparats (estat ready) a esperar que el planificador li assigni la CPU.

b) En un sistema de gestió de memòria basat en paginació sota demanda, seria possible que, en un mateix moment, una mateixa direcció lògica pugui ser vàlida per a dos processos diferents?

La mateixa direcció lògica no pot ser vàlida per a dos processos diferents pel fet que cada procés té el seu espai d'adreces lògiques independents. Si que pot ocórrer que cada un d'ells accedeixi a les mateixes adreces lògiques, però la seva validesa o no dependrà de la taula de pàgines de cada procés.

c) A Linux, un usuari pot accedir a arxius dels que no es propietari? En cas afirmatiu, indiqueu com.

Si que es pot accedir als arxius dels que no és propietari sempre que l'arxiu ho permeti mitjançant els permisos del grup d'usuaris o d'altres usuaris.

d) A Linux, com a màxim, quant de temps pot estar un procés en estat *zombie*? Podria ocórrer que un sistema Linux arribés a saturar-se a causa dels processos *zombies*?

La durada màxima d'un procés zombi és fins que finalitzi el procés pare, en aquest moment si el pare no realitza el wait els processos fills zombis, són adoptats pel procés init i eliminats del sistema.

A Linux no pot ocórrer que el sistema es saturi de processos zombi, ja que tan aviat com un procés finalitza sense fer el wait dels fills, aquests fills que podrien quedar en un estat de zombie indefinit, ja que són adoptats pel procés init que realitza el wait perquè puguin ser eliminats del sistema correctament.

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

2. Memòria [2,5 punts]

Sigui un sistema de gestió de memòria basat en paginació sota demanda on les pàgines tenen una mida de 4KBytes, les adreces lògiques són de 16 bits i l'espai físic és de 32 KBytes.

Sobre aquest sistema es creen dos processos.

- Procés 1: el seu fitxer executable determina que l'àrea de codi ocupa una pàgina, que les dades inicialitzades n'ocupen una, les no inicialitzades dues i la pila una.
- Procés 2: el seu fitxer executable determina que l'àrea de codi ocupa una pàgina, que les dades inicialitzades n'ocupen una, les no inicialitzades una i la pila una.

Es demana:

a) Estimeu la mida del fitxer executable corresponent al procés 1.

El fitxer executable conté el codi i el valor inicial de les dades inicialitzades. En aquest cas, tenim una pàgina de codi i una de dades inicialitzades per tant, aproximadament, l'executable ocuparà la mida corresponent a dues pàgines, és a dir, uns 8KB.
Per a una estimació més precisa caldria afegir la mida de les capçaleres de l'executable i caldria descomptar la fragmentació interna que pugui existir a la darrera pàgina de codi i de dades inicialitzades.

b) Suposant que les pàgines es carreguen a memòria física tal i com indica el diagrama següent, indiqueu quin serà el contingut de les taules de pàgines de tots dos processos (podeu contestar sobre el diagrama de l'enunciat).

c) Suposant que el procés en execució és el procés 1, indiqueu quines seran les adreces físiques corresponents a les següents adreces lògiques: `0xE9AB` i `0xF452`. Variaria la resposta si el procés en execució fos el procés 2? En cas afirmatiu, indiqueu el motiu i com canviaria.

Primer cal descompondre les adreces lògiques en identificador de pàgina i desplaçament.
Com la mida de pàgina és de 4KB (2^12 bytes), calen 12 bits per a codificar el desplaçament dins de la pàgina, és a dir, tres dígits hexadecimals. La resta de bits (4) seran l'identificador de pàgina. Per tant, el primer dígit hexadecimal ens indica l'identificador de pàgina lògica i la resta de dígits indiquen el desplaçament dins la pàgina.

Les traduccions serien:

@L	@F Procés 1	@F Procés 2
0xE9AB	Excepció adreça invàlida	Excepció adreça invàlida
0xF452	0x3452	Excepció: fallada de pàgina

Són diferents a cada procés perquè cada procés té una taula de pàgines pròpia.

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

Procés 1

Espai lògic

pàg. lòg.	
0x0	C1-1
0x1	D1-1
0x2	D1-2
0x3	D1-3
0x4	-
0x5	-
0x6	-
0x7	-
0x8	-
0x9	-
0xA	-
0xB	-
0xC	-
0xD	-
0xE	-
0xF	P1-1

Taula de pàgines

	valid	present	frame
0x0			
0x1			
0x2			
0x3			
0x4			
0x5			
0x6			
0x7			
0x8			
0x9			
0xA			
0xB			
0xC			
0xD			
0xE			
0xF			

Procés 2

Espai lògic

pàg. lòg.	
0x0	C2-1
0x1	D2-1
0x2	D2-2
0x3	-
0x4	-
0x5	-
0x6	-
0x7	-
0x8	-
0x9	-
0xA	-
0xB	-
0xC	-
0xD	-
0xE	-
0xF	P2-1

Taula de pàgines

	valid	present	frame
0x0			
0x1			
0x2			
0x3			
0x4			
0x5			
0x6			
0x7			
0x8			
0x9			
0xA			
0xB			
0xC			
0xD			
0xE			
0xF			

Espai físic

frame	
0x0	D2-2
0x1	
0x2	D1-1
0x3	P1-1
0x4	D2-1
0x5	
0x6	C2-1
0x7	

Àrea de swap

C1-1	D1-2
P2-1	D1-3

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

Procés 1

Espai lògic

pàg. lòg.	
0x0	C1-1
0x1	D1-1
0x2	D1-2
0x3	D1-3
0x4	-
0x5	-
0x6	-
0x7	-
0x8	-
0x9	-
0xA	-
0xB	-
0xC	-
0xD	-
0xE	-
0xF	P1-1

Taula de pàgines

	valid	present	frame
0x0	1	0	
0x1	1	1	0x2
0x2	1	0	
0x3	1	0	
0x4	0		
0x5	0		
0x6	0		
0x7	0		
0x8	0		
0x9	0		
0xA	0		
0xB	0		
0xC	0		
0xD	0		
0xE	0		
0xF	1	1	0x3

Procés 2

Espai lògic

pàg. lòg.	
0x0	C2-1
0x1	D2-1
0x2	D2-2
0x3	-
0x4	-
0x5	-
0x6	-
0x7	-
0x8	-
0x9	-
0xA	-
0xB	-
0xC	-
0xD	-
0xE	-
0xF	P2-1

Taula de pàgines

	valid	present	frame
0x0	1	1	0x6
0x1	1	1	0x4
0x2	1	1	0x0
0x3	0		
0x4	0		
0x5	0		
0x6	0		
0x7	0		
0x8	0		
0x9	0		
0xA	0		
0xB	0		
0xC	0		
0xD	0		
0xE	0		
0xF	1	0	

Espai físic

frame	
0x0	D2-2
0x1	
0x2	D1-1
0x3	P1-1
0x4	D2-1
0x5	
0x6	C2-1
0x7	

Àrea de swap

C1-1	D1-2
P2-1	D1-3

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

3. Processos [2,5 punts]

a) Indiqueu quin és el comportament dels següents programes (jerarquia de processos creada, missatges escrits per sortida estàndard/pipes/fitxers i en quin ordre,...). Podeu assumir que cap crida al sistema retornarà error.

```
/* A */

main()
{
  int p;

  write(1, "Hello\n", 6);
  p = fork();
  if (p>0) {
    write(1, "Bye1\n", 5);
  }
  else {
    wait(NULL);
    write(1, "Bye2\n", 5);
  }
  exit(0);
}
```

```
/* B */

main()
{
  int p[2], j

  write(1, "H
  pipe(p);
  j = fork();
  if (j == 0)
    write(1,
    write(p[1
  }
  else {
    close(p[1
    while (re
      write(1
```

a) El procés inicial escriu “Hello\n”. A continuació es crea un procés fill. El procés pare escriu “Bye1\n” i mor. El procés fill invoca a wait però, al no tenir cap fill, la crida retorna immediatament; a continuació escriu “Bye2” i mor. No podem saber en quin ordre s’escriuran els dos missatges.

b) El procés inicial escriu “Hello\n”. A continuació crea una pipe i un procés fill. El procés fill escriu “Bye1” per la sortida estàndar, escriu “abc” a la pipe i mor. El procés pare tanca el canal d’escriptura de la pipe i entra en un bucle de lectura de la pipe. Les tres primeres iteracions es sincronitzen amb l’escriptura del procés fill amb el que s’escriu tres vegades per la sortida estàndar el missatge “Looping\n”. A la quarta iteració el procés surt del bucle perquè la pipe és buida i no existeix cap canal d’escriptura obert sobre la pipe. El procés pare escriu “Bye2” i mor. El procés fill serà adoptat pel procés init i podrà ser eliminat del sistema.

c) El procés tanca la sortida estàndar i obre el fitxer file que, assumint que el canal 0 està ocupat, serà accessible pel canal 1. A continuació carrega l’executable “ls” i l’executa amb el que es mostra per la sortida estàndar (el fitxer file) la llista de fitxers del directori actual.

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

b) Escriviu un programa que creï tres processos fills F1, F2 i F3 on:

- F1 passarà a executar el programa `prog1` havent-li redireccionat l'entrada estàndard al fitxer `file_in.txt` i la sortida estàndard al fitxer `file_out.txt`
- Quan el procés F1 hagi acabat, F2 passarà a executar `prog2` i F3 passarà a executar `prog3` **concurrentment**.. En tots dos casos havent redireccionat l'entrada estàndard al fitxer `file_out.txt`. Respecte a la sortida estàndard, F2 la tindrà redireccionada al fitxer `file_out2.txt` i F3 al `file_out3.txt`.
- Cal que indiqueu tots els paràmetres a les crides al sistema i que tracteu el valor de retorn coherentment. No cal indicar els includes ni fer el tractament d'errors a les crides al sistema. Podeu assumir que els fitxers d'entrada ja existeixen i que els de sortida no existeixen.

```
main()
{
    if (fork() == 0) {
        close(0);
        open("file_in.txt", O_RDONLY);
        close(1);
        open("file_out1.txt", O_WRONLY);
        execl("prog1", "prog1", NULL);
    }

    wait(NULL);

    if (fork() == 0) {
        close(0);
        open("file_out1.txt", O_RDONLY);
        close(1);
        open("file_out2.txt", O_WRONLY);
        execl("prog2", "prog2", NULL);
    }

    if (fork() == 0) {
        close(0);
        open("file_out1.txt", O_RDONLY);
        close(1);
        open("file_out2.txt", O_WRONLY);
        execl("prog3", "prog3", NULL);
    }

    wait(NULL);
    wait(NULL);
}
```

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

4. Concurrència [2.5 punts]

Tenim una aplicació concurrent formada per N fils d'execució. Cada un d'aquests fils d'execució rep com a paràmetre un identificador que identifica l'ordre lògic en què s'ha creat el procés (des d'1 fins a N). De moment, els fils, únicament imprimeixen el seu identificador per pantalla:

```
Thread(int id)
{
   char msg;

   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));
}
```

Utilitzant les següents operacions de semàfors:

- `sem_init(semaphore s, int valor)`. Inicialitza el semàfor *s* amb *valor* instàncies inicials (valor inicial).
- `sem_wait(semaphore s)`. Demana una instància del semàfor *s*. Espera que el valor del semàfor sigui més gran que 0 i quan ho és el decrementa de forma atòmica.
- `sem_wait_mult(semaphore s, int n)`. Demana *n* instàncies del semàfor *s*. Espera que el valor del semàfor sigui més gran que *n*-1 i quan ho és el decrementa en *n* de forma atòmica.
- `sem_signal(semaphore s)`. S'incrementa de forma atòmica el valor del semàfor.
- `sem_signal_mult(semaphore s, int n)`. Augmenta de forma atòmica el valor del semàfor en *n*.

Es demana:

a) Assumint que es creen N fils, garantiu amb semàfors que els fils amb identificador sigui superior a 10 no puguin mostrar el seu missatge fins que els primers N fils hagin mostrat el seu. Implementeu el codi del fil 1 i del fil 11.

```
Semaphore SemOver10, SemMutex;
intcount=0;

sem_init(SemOver10,0);
sem_init(SemMutex,1);
```

```
Thread1(int id)
{
   charmsg;

   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));

   sem_wait(SemMutex]);
   count++;
   if (count==10)
        sem_signal(SemOver10, N-10);
   sem_signal(SemMutex]);
}
```

```
Thread11(int id)
{
   charmsg;

   sem_wait(SemOver10);

   sprintf(msg,"Thread %d.\n",id);
   write(1,msg,strlen(msg));

}
```

Examen 2018/19-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	19/01/2019	09:00

b) Assumiu ara que tenim fils que escriuen el seu nombre per pantalla (*ThreadNumber*) i fils que escriuen una lletra per pantalla (*ThreadCharacter*). Modifiqueu el codi d'aquests dos fils, de manera que es garanteixi amb semàfors que si està escrivint en una línia nombres, els fils caràcter no puguin escriure i viceversa. L'últim fil que escrigui en una línia, si cap altre fil del mateix tipus vol escriure, finalitzarà la línia escrivint un retorn de carro (salt de línia).

Una possible sortida vàlida seria la següent:

1 5 7 2 6
A E D
4 3 8
F
9

```
Semaphore SemMutexN , SemMutexC, SemScreen;
Shared int numbers=0, characters=0;

sem_init(SemMutexN,1);
sem_init(SemMutexC,1);
sem_init(SemScreen,1);
```

```
ThreadNumber()
{
   sem_wait(SemMutexN);
   numbers++;
   if (numbers==1)
      sem_wait(SemScreen);
   sem_signal(SemMutexN);

   sprintf(msg,"%d ",id);
   write(1,msg,strlen(msg));

   sem_wait(SemMutexN);
   numbers--;
   if (numbers==0) {
      sem_signal(SemScreen);
      write(1,"\n",1);
   }
   sem_signal(SemMutexN);
}
```

```
ThreadCharacter()
{
   sem_wait(SemMutexC);
   characters++;
   if (characters==1)
      sem_wait(SemScreen);
   sem_signal(SemMutexC);

   sprintf(msg,"%c ",'A'+id);
   write(1,msg,strlen(msg));

   sem_wait(SemMutexC);
   characters--;
   if (characters==0) {
      sem_signal(SemScreen);
      write(1,"\n",1);
   }
   sem_signal(SemMutexC);
}
```