Espai grapa

Examen 2017/18-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

05.566 13 01 18 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 en la qual estàs matriculat.**
- **Només has d'enganxar una etiqueta d'estudiant a l'espai corresponent d'aquest full.**
- **No es poden adjuntar fulls addicionals.**
- **No es pot realitzar la prova en llapis ni en retolador gruixut.**
- **Temps total:** 2 **h.**
- **En cas que els estudiants puguin consultar algun material durant l'examen, quin o quins materials poden consultar?**
  Un full mida foli/DIN A-4 amb anotacions per les dues cares
- **Valor de cada pregunta:** 2,5 punts
- **En cas que hi hagi preguntes tipus test: Descompten les respostes errònies?** NO **Quant?**

- **Indicacions específiques per a la realització d'aquest examen:**
  Poden portar calculador per realitzar l'examen.

Enunciats

Examen 2017/18-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

1. Teoria [2.5 punts]

Contesteu **justificadament** les següents preguntes:

a) Donat el següent estat inicial:

```
int x, y;
sem_t semA;
x = 1;
y = 3;
sem_init(semA, 0, 1);  // S'inicialitza el semàfor SemA a 1 (una instància)
```

Enumerar els possibles valors de x per la següent execució, després que els dos fils (Thread1 i Thread2) hagin acabat o es bloquegin. Indicar també si algun dels fils es bloqueja indefinidament i perquè.

```
Thread1:
   sem_wait(SemA);
   x = x+1;
   sem_signal(SemA);
```

```
Thread2:
   sem_wait(SemA);
   x = x+2;
   sem_signal(SemA);
```

En aquest codi tenim dos fils que accedeixen a la variable compartida x. Tots dos fils implementen una mateixa secció crítica (mitjançant el semàfor A) pel que mentre un dels dos fils és a la SC l'altre s'ha d'esperar.

En aquest cas l'únic possible valor que podem obtenir de la variable x seria 4.

Cap fil es bloqueja indefinidament.

b) ¿Indiqueu quants processos, pipes i redireccions necessita l'intèrpret de comandes per executar la següent comanda?

```
$ grep nom < dades1.txt > dades2.txt
```

Es necessita un procés per executar la comanda grep. No s'utilitza cap pipe i es necessita realitzar 2 redireccions: redirigir la stdout del procés grep al fitxer dades2.txt i redirigir el fitxer dades1.txt a la stdin del procés grep.

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

c) En un sistema paginat, sense memòria virtual, l'espai que un procés ocupa en memòria, just després de la seva càrrega, es pot veure afectat per la fragmentació interna?

Cert. L'espai ocupat per un procés just després de la seva càrrega és igual a l'espai ocupat pel codi i les dades estàtiques. Com que no tenim memòria virtual totes les pagines han d'estar carregats a memòria. En paginar dividim l'espai lògic del procés en pàgines d'igual magnitud, i cada pàgina es carrega en un frame diferent. Difícilment l'espai ocupat pel codi i les dades estàtiques seran un múltiple exacte de la mida d'una pàgina; per això el darrer frame de cada fragment sempre tindrà posicions no usades: és el fenomen de la fragmentació interna. En mitjana les posicions no usades en el darrer frame són la meitat de la mida d'una pàgina o frame.

d) Es veritat que l'ús d'una operació d'E/S bloquejadora fa que durant aquesta operació (per exemple un read) el sistema quedi bloquejat?

Fals. Qui queda bloquejat és el procés que ha iniciat l'operació d'E/S. De fet el que passa és que el SO el porta a la cua de processos bloquejats, i un dels processos dels que estan Preparats li dóna el control de la CPU (el passa a Execució).

Examen 2017/18-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

2. Processos [2.5 punts]

Contesteu les següents preguntes:

a) Indiqueu un motiu que pugui provocar que un procés passi d'estat Run a estat Wait.

Quan un procés invoca una crida al sistema bloquejant, com ara la lectura sobre una pipe buida, el procés passa de l'estat Run a l'estat Wait.

b) Indiqueu quin serà el resultat d'executar el següent programa (jerarquia de processos creada, informació mostrada per la sortida estàndar). Podeu assumir que cap crida al sistema retornarà error.

```
main()
{ int a;
  char s[80];

  if (fork() == 0) a=0; else a=1;

  sprintf(s, "a=%d\n", a, b);
  write(1, s, strlen(s));

  execl("/bin/ls", "ls", NULL);
  write(1, s, strlen(s));
}
```

La primera sentència crea un procés fill; el fill tindrà a=0 i el pare a=1.
Tots dos processos imprimeixen el seu valor de "a" (no podem saber en quin ordre) i després tots dos processos invoquen execl i passen a executar "ls"; per tant, apareix dos cops la llista de fitxers del directori actual.
El write posterior no s'executarà perquè, assumint que l'exec no retornarà error), la crida execl no retorna.

c) Escriviu un programa que cada cop que l'usuari premi un salt de línia, el programa indiqui quants processos vius hi ha al sistema mitjançant la comanda "ps –aux | grep –c" No és necessari que implementeu el tractament d'errors a les crides al sistema. Cal implementar el programa utilitzant crides al sistema Unix, no és permès utilitzar rutines de biblioteca com ara system().

```
void printPsGrep()
{
  int fd[2];

  if (pipe(fd) < 0) error("pipe");

  switch (fork ())
    {
    case -1:
      error ("fork");
```

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

```
    case 0:
      close(1); dup(fd[1]); close(fd[0]); close(fd[1]);
      execlp ("ps", "ps", “-aux”, NULL);
      error ("execl");
    }

  switch (fork ())
    {
    case -1:
      error ("fork");

    case 0:
      close(0); dup(fd[0]); close(fd[0]); close(fd[1]);
      execlp ("grep", "grep", “-c”, NULL);
      error ("execl");
    }

  close(fd[0]); close(fd[1]);

  wait(NULL); wait(NULL);
}

int main (int argc, char *argv[])
{
  char c;
  int r;

  while ((r = read(0, &c, 1)) > 0)
    {
      if (c=='\n')
        printPsGrep();
    }

  if (r < 0)
    error("read");

  return 0;
}
```

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

d) Què és un procés zombie/defunct?

És un procés que ja ha invocat la crida al sistema exit, per tant ja és mort, però que el seu pare encara no ha invocat la crida wait per tenir constància de la seva finalització.

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

3. Memòria [2.5 punts]

Sigui un sistema de gestió de memòria basat en paginació sota demanda on les pàgines tenen una mida de 64KBytes, les adreces lògiques són de 20 bits i l'espai físic és de 512 KBytes.

Sobre aquest sistema es creen dos processos.

- Procés 1: el seu fitxer executable determina que el codi ocuparà dues pàgines, que les dades inicialitzades n'ocupen una, les no inicialitzades dues i la pila també dues.
- Procés 2: el seu fitxer executable determina que les àrees de codi i les dades inicialitzades ocuparan una pàgina cadascuna, que no existeixen dades no inicialitzades i que la pila ocuparà dues pàgines.

Es demana:

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

El fitxer executable d'un procés conté el codi i el valor inicial de les dades inicialitzades. En aquest cas, tenim dos pàgines de codi i una de dades inicialitzades per tant, aproximadament, l'executable ocuparà la mida corresponent a tres pàgines, és a dir, uns 192KB.

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.

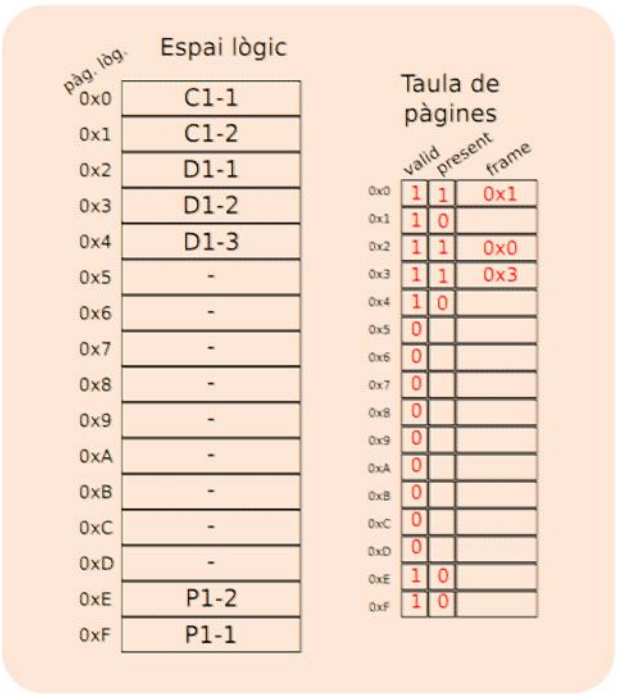
Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

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).

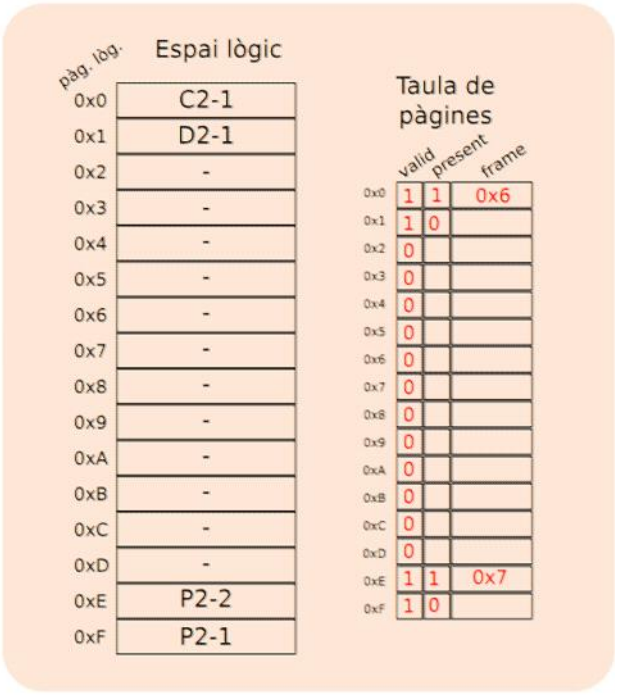
Examen 2017/18-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

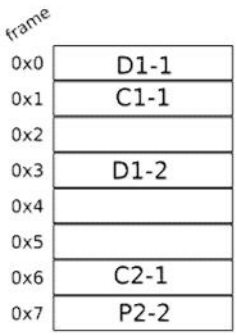
Procés 1

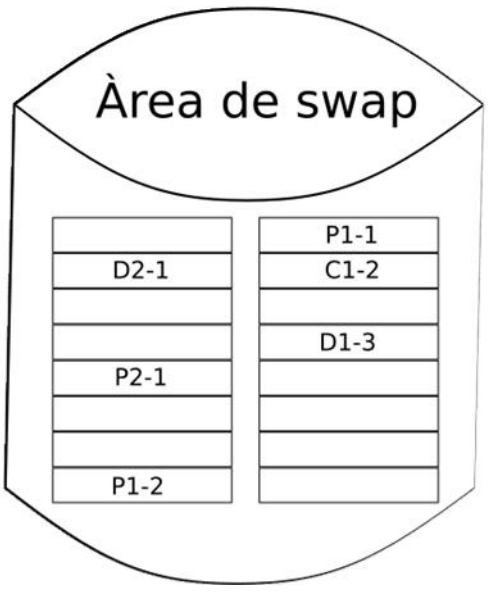


Procés 2



Espai físic





Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

c) Suposant que el procés en execució és el procés 1, indiqueu quines seran les adreces físiques corresponents a les següents adreces lògiques: 0x0A123 i 0x4B342. Variaria la resposta si el procés en execució fos el procés 2? En cas afirmatiu, indiqueu el motiu i com canviaria.

Primer cal descomposar les adreces lògiques en identificador de pàgina i desplaçament.

Com la mida de pàgina és de 64KB (2^16 bytes), calen 16 bits per a codificar el desplaçament dins de la pàgina, és a dir, quatre 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
0x0A123	0x1A123	0x6A123
0x4B342	Fallada de pàgina	Excepció: Adreça invàlida

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

Examen 2017/18-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

4. Concurrència [2.5 punts]

La PAC2 de l'assignatura versava sobre el problema de com controlar l'accés a un pont estret per part de conjunt de cotxes, simulats mitjançant fils d'execució, que volien creuar-ho en ambdues direccions. En tot moment, només es permet creuar cotxes en un sentit, havent d'esperar els cotxes de l'altra direcció fins que se'ls concedeixi el torn.

a) Modificar el codi de la primera solució de l'aplicació concurrent del pont estret de la 2a PAC, perquè es generi un deadlock. Indicar la seqüència exacta d'operacions que generarà el deadlock.

```
sem_t Bridge;
sem_init(Bridge, 0, 1);
```

```
CarCrossEast()
{
   sem_wait(Bridge);

   CrossingBridge();

   sem_signal(Bridge);
}
```

```
CarCrossWest()
{
   sem_wait(Bridge);

   CrossingBridge();

   sem_signal(Bridge);
}
```

Codi 1. Solució que només permet creuar un cotxe simultàniament.

Amb aquesta sincronització es pot produir un *deadlock*, si un fil1 que està executant la funció *`CarCrossEast()`* adquireix la instància del semàfor *`Bridge1`* i abans d'adquirir el següent semàfor (*`Bridge2`*), s'executa un segon fil que adquireix l'instància del semàfor *`Bridge2`*. En aquesta situació, el fil1 té el semàfor *`Bridge1`*, però es bloqueja per que necessita també el semàfor *`Bridge2`* que està assignat al fil2 que està bloquejat perquè necessita el semàfor *`Bridge1`* que està assignat al fil1. Per tant, apareix l'espera circular i cap dels dos fils podrà completar la seva execució (*Deadlock*)

```
sem_t Bridge1, Bridge1;
sem_init(Bridge1, 0, 1);
sem_init(Bridge2, 0, 1);
```

```
CarCrossEast()
{
   sem_wait(Bridge1);
   sem_wait(Bridge2);

   CrossingBridge();

   sem_signal(Bridge1);
   sem_signal(Bridge2);
}
```

```
CarCrossWest()
{
   sem_wait(Bridge2);
   sem_wait(Bridge1);

   CrossingBridge();

   sem_signal(Bridge2);
   sem_signal(Bridge1);
}
```

Examen 2017/18-1

Assignatura	Codi	Data	Hora inici
Sistemes operatius	05.566	13/01/2018	15:30

b) Ens proposen la següent solució per controlar mitjançant semàfors que com a màxim només pot haver-hi 3 cotxes simultàniament al pont.

```
Shared int CarsEast=0, CarsWest=0;
sem_t Bridge, MutexEast, MutexWest;

sem_init(Bridge, 0, 1);
sem_init(MutexEast, 0, 1);
sem_init(MutexWest, 0, 1);
```

```
CarCrossEast()
{
   sem_wait(MutexEast);
   CarsEast++;
   if ( CarsEast==1 ||
        CarsEast>3 )
Inst1a:  sem_wait(Bridge);
   sem_signal(MutexEast);

   CrossingBridge();

   sem_wait(MutexEast);
   CarsEast--;
   if ( CarsEast==0 ||
        CarsEast>2 )
Inst2a:  sem_signal(Bridge);
   sem_signal(MutexEast);
}
```

```
CarCrossWest()
{
   sem_wait(MutexWest);
   CarsWest++;
   if ( CarsWest==1 ||
        CarsWest>3 )
Inst1b:  sem_wait(Bridge);
   sem_signal(MutexWest);

   CrossingBridge();

   sem_wait(MutexWest);
   CarsWest--;
   if ( CarsWest==0||
        CarsWest>2)
Inst2b:  sem_signal(Bridge);
   sem_signal(MutexWest);
}
```

Indicar per que aquesta solució no és correcta. Justifiqueu la resposta.

Aquesta solució intenta que quan hi hagi més de 3 cotxes creuant el pont (`CarsEast>3`), aquests es quedin bloquejats en executar la instrucció 1a (`sem_wait(Bridge);`) ja que d'aquest semàfor només hi ha una instància disponible.

D'altra banda, cada vegada que el pont està a la seva màxima capacitat i un dels cotxe ha creuat el pont (`CarsEast>2`), s'allibera una instància addicional del semàfor *Bridge* (en la instrucció 2a) perquè un altre dels cotxes pugui creuar en el mateix sentit.

El problema d'aquesta solució és que no et garanteix que la instància que allibera un cotxe que ha acabat de creuar cap a l'Est (*instr2a*) no la utilitzi un cotxe per creuar en sentit contrari (executant la instrucció 1b). Amb el que tindríem cotxes creuant en sentit contrari, i per tant la solució no seria correcta.