Recuperatorio 2015

sábado, 20 de mayo de 2023 03:45 a. m.

Sistemas Operativos
Primer Recuperatorio Parcial 1

UTN FRD
Ingeniería en Sistemas de Información
Dpto. Ingeniería en Sistemas de Información
Área: Computación

Apellido y Nombres …….

1.- El siguiente código intenta representar la solución a la siguiente problemática:
Un proceso crea un conjunto de procesos hijos al mismo nivel y le asigna a cada uno una tarea determinada, cada proceso hijo debe indicar a su proceso padre un estado de finalización de la tarea, estado 1 tarea resuelta con éxito, estado 0 tarea no resuelta. Suponemos que el fork() no falla.(2,5)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int tarea(int);
int main(int  argc , char *argv[]) {
    pid_t pid , pidTer;
    int status  = 0 ;
    int np = atoi(argv[1]);
    while(np)  {
        pid = fork();
        np -- ;
        if (pid == 0 ) exit(tarea(getpid()));
    }
   // Aquí el alumno tiene que escribir el código para que no existan
procesos zombies ni
   //huérfanos indicando para cada hijo PID y ESTADO DE TERMINACION.
…
…
…
…
    exit(0);
}

int tarea(int num) {
  sleep(1);
  printf("tarea %d \n",num);
  return num % 2;
}
```

2.- Reescribir las funciones de los hilos.
Reescriba las funciones de los hilos para que el siguiente código en lenguaje C permita que que tres hilos se sincronicen para escribir en pantalla “soy hilo 1” “soy hilo 2” “soy hilo 1” “soy hilo 3” infinitamente. No puede usar el concepto de variable bandera. En la pantalla se visualizará infinitamente:(2,5)
Soy Hilo 1
Soy Hilo 2
Soy Hilo 1
Soy Hilo 3
Soy Hilo 1
Soy Hilo 2
Soy Hilo 1
Soy Hilo 3
….

```
#include <stdio.h>
```

Sistemas Operativos
Primer Recuperatorio Parcial 1

UTN FRD
Ingeniería en Sistemas de Información
Dpto. Ingeniería en Sistemas de Información
Área: Computación

Apellido y Nombres …….

```
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void Hilo1(void);
void Hilo2(void);
void Hilo3(void);

pthread_mutex_t  m1 = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_t  m2 = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_t  m3 = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_t  m4 = PTHREAD_MUTEX_INITIALIZER ;

int main(void) {
  pthread_t h1 ;
  pthread_t h2 ;
  pthread_t h3 ;

  pthread_mutex_lock(&m3);
  pthread_mutex_lock(&m4);

  pthread_create(&h1,NULL,(void *)Hilo1,NULL);
  pthread_create(&h2,NULL,(void *)Hilo2,NULL);
  pthread_create(&h3,NULL,(void *)Hilo3,NULL);

  pthread_join(h1,NULL);
  printf("Nunca Termina\n");
  exit(1);
}

void Hilo1(void) {   while(1)   printf("Soy Hilo 1 \n "); }

void Hilo2(void) {   while(1)  printf("Soy Hilo 2 \n "); }

void Hilo3(void) {   while(1)  printf("Soy Hilo 3 \n "); }
```

3.- Señales: Un proceso padre se sincroniza con un proceso hijo usando señales, comienza el padre escribiendo “Soy el Padre” luego escribe el hijo “Soy el Hijo” y se intercalan infinitamente. Se debe completar el código para dicho objetivo y además para que los procesos padre e hijo nunca puedan recibir ninguna señal que no sean las necesarias para la sincronización.**(2,5)**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h >

void f_signalPadre(void) ;
void f_signalHijo(void) ;

int main(void) {
  pid_t pid;
  pid = fork();
```

```
  pid = fork();
  if (pid == 0) {
        signal(SIGUSR2,f_signalHijo) ;
```

Sistemas Operativos
Primer Recuperatorio Parcial 1

UTN FRD
Ingeniería en Sistemas de Información
Dpto. Ingeniería en Sistemas de Información
Área: Computación

Apellido y Nombres …….

```
        while(1) {

        }
  } else {
       signal(SIGUSR1,f_signalPadre) ;
       while(1) {

       }
  }
  exit(0);
}

void f_signalPadre(void) {
     printf("Soy el Padre \n");
}
void f_signalHijo(void) {
     printf("Soy el Hijo \n");
}
```

4) Obtener el tiempo medio de espera y el porcentaje de sobrecarga del sistema operativo para la siguiente definición de procesos para un algoritmo por prioridades apropiativo. Las interrupciones son no enmascarables. Prioridad de PB mayor a la prioridad de PA. Indicar con "E" ejecución, con "D" disco, con "C" cinta y con "S" suspendido si es necesario. Los cambios de contexto duran 10 ms cada uno.(2,5)

PA	**CPU 30 ms., E/S disco 20ms, CPU 40 ms, E/S cinta 20 ms., CPU 30ms.**
PB	**CPU 20 ms., E/S disco 20 ms., CPU 20 ms, E/S cinta 20 ms, CPU 30ms.**

Tiempo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
S.O.																																	
Proceso PA																																	
Proceso PB																																	
Interrupción																																	
Cola de listos																																	

5) ¿Qué es una interrupción? (2,5)
6) ¿Qué diferencia hay entre "trap" e interrupción? (2,5)
7) ¿Qué sucede si un proceso queda indefinidamente dentro de una sección crítica? (2,5)
8) ¿Qué impacto tiene el tamaño de la "rodaja de tiempo" (quatum) usado en la política de "Turno Rotatorio" (Round Robin) en el rendimiento del SO? (2,5)

Criterio de Aprobación: obtener 6/10 puntos en la teoría (preguntas 5 a 8) y obtener 6/10 puntos en la práctica (preguntas 1 a 4). Solo se considera un punto correcto si todos sus items se respondieron correctamente.

Sistemas Operativos
Primer Recuperatorio Parcial 1

UTN FRD
Ingeniería en Sistemas de Información
Dpto. Ingeniería en Sistemas de Información
Área: Computación

Apellido y Nombres …….

SOLUCIONES
1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int tarea(int);

int main(int  argc , char *argv[])
{
   pid_t pid , pidter;
   int status  = 0 ;
   int np = atoi(argv[1]);
   while(np)
   {
      pid = fork();
      np--;
      if (pid == 0 ) exit(tarea(getpid()));
   }
   while((pidter = wait(&status)) != -1 )
    {
      printf("Hijo %d termino  ",pidter) ;
      printf("retorno %d\n",status/256);
    }
   exit(0);
}

int tarea(int num)
{
 sleep(1);
 printf("tarea %d \n",num);
 return num % 2;
}
```

2

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void Hilo1(void);
```

void Hilo2(void);

Sistemas Operativos
Primer Recuperatorio Parcial 1

UTN FRD
Ingeniería en Sistemas de Información
Dpto. Ingeniería en Sistemas de Información
Área: Computación

Apellido y Nombres …….
void Hilo3(void);

```
pthread_mutex_t  m1 = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_t  m2 = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_t  m3 = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_t  m4 = PTHREAD_MUTEX_INITIALIZER ;

int main(void)
{
 pthread_t h1 ;
 pthread_t h2 ;
 pthread_t h3 ;

 pthread_mutex_lock(&m3);
 pthread_mutex_lock(&m4);

 pthread_create(&h1,NULL,(void *)Hilo1,NULL);
 pthread_create(&h2,NULL,(void *)Hilo2,NULL);
 pthread_create(&h3,NULL,(void *)Hilo3,NULL);

  pthread_join(h1,NULL);
  printf("Nunca Termina\n");
  exit(1);
}

void Hilo1(void)
{
 while(1)
    {
      pthread_mutex_lock(&m1);
      printf("Soy Hilo 1 \n ");
      sleep(1);
      pthread_mutex_unlock(&m4);
    }
}

void Hilo2(void)
{
 while(1)
    {
      pthread_mutex_lock(&m2);
      pthread_mutex_lock(&m4);
      printf("Soy Hilo 2 \n ");
      sleep(1);
      pthread_mutex_unlock(&m1);
      pthread_mutex_unlock(&m3);
    }
}
```

Sistemas Operativos
Primer Recuperatorio Parcial 1

UTN FRD
Ingeniería en Sistemas de Información
Dpto. Ingeniería en Sistemas de Información
Área: Computación

Apellido y Nombres …….

```
void Hilo3(void)
{
 while(1)
    {
      pthread_mutex_lock(&m3);
      pthread_mutex_lock(&m4);
      printf("Soy Hilo 3 \n ");
      sleep(1);
      pthread_mutex_unlock(&m1);
      pthread_mutex_unlock(&m2);
    }
}
```

3

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void f_signalPadre(void) ;
void f_signalHijo(void) ;

int main(void)
{
  pid_t pid ;
  pid = fork();
  if (pid == 0 )
     {
       sigset_t seth ;
       sigfillset(&seth);
       sigdelset(&seth,SIGUSR2);
       signal(SIGUSR2,(void *)f_signalHijo) ;
       sigprocmask(SIG_BLOCK,&seth,NULL);
       while(1)
          {
            sleep(1);
            kill(getppid(),SIGUSR1);
            pause();

          }
     }
  else
     {
       sigset_t setp ;
       sigfillset(&setp);
```

Sistemas Operativos
Primer Recuperatorio Parcial 1

UTN FRD
Ingeniería en Sistemas de Información
Dpto. Ingeniería en Sistemas de Información
Área: Computación

Apellido y Nombres …….

```
    sigdelset(&setp,SIGUSR1);
    signal(SIGUSR1,(void *)f_signalPadre) ;
    sigprocmask(SIG_BLOCK,&setp,NULL);
    while(1)
       {
        pause();
        kill(pid,SIGUSR2);
        sleep(1);
       }
  }
  exit(0);
}

void f_signalPadre(void)
{
  printf("Soy el Padre \n");
}

void f_signalHijo(void)
{
  printf("Soy el Hijo \n");
}
```

5 ¿Qué es una interrupción?
Mecanismo por el cual otros componentes (ej. módulos E/S) pueden interrumpir la normal ejecución de la CPU.

6 ¿Qué diferencia hay entre “trap” e interrupción?
Son conceptos similares, pero no no idénticos, un trap es una forma de interrupción pero que se origina dentro del propio proceso interrumpido, asociado a una condicion de error o excepción. Mientras que una interrupción es algo externo al proceso en ejecución, por ejemplo, puede ser que se haya agotado la rodaja de tiempo asignada para la ejecucion del mismo (time slice) entonces se produce una interrupción de reloj (clock).

7 ¿Qué sucede si un proceso queda indefinidamente dentro de una sección crítica?
El resto de los procesos -que también ejecutan esta sección crítica- quedarán bloqueados intentando entrar en ella.

8 ¿Qué impacto tiene el tamaño de la "rodaja de tiempo" (quatum) usado en la política de "Turno Rotatorio" (Round Robin) en el rendimiento del SO?
Muy chico implica mayor sobrecarga de cpu y muy grande implica que funcione como FCFS.

Criterio de Aprobación: **obtener 6/10 puntos en practica y teoria. 2.5 puntos por ejercicio.**
Criterio de Calificación % aprobación promedio

60% 4 70% 6 80% 8 90% 9 100% 10