


Parcial 13-07-2015

sábado, 20 de mayo de 2023 03:19 a. m.



Primer Examen Parcial 13/07/15

UTN FRD
Ingeniería en Sistemas de Información
Área: Computación

① Proceso \Rightarrow Crea un qto de procesos
Asigna a c/uno una tarea.
c/ proceso debe indicar a su proceso creador

ESTADO 1 \Rightarrow TAREA RESUELTA ESTADO 0 \Rightarrow TAREA NO RESUELTA

1.6

pid = fork()

ÁRBOL
PROCESOS

\rightarrow el wait contiene el estado de finalización del proceso


wait(int *status)

pidter = wait(status1)
pidter = wait(status2)
pidter = wait(status3)

debo agregar
a c/uno

$\&$ \Rightarrow Cuanto usamos & adelante de una variable se busca obtener la dirección de memoria de la variable

$*$ \Rightarrow indica que la variable es un puntero



Sistemas Operativos
Primer Examen Parcial 13/07/15

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

Apellido y Nombre:

```

Fin
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
void hilo2(void);
void hilo1(void);
int main(void) {
    pthread_t h1;
    pthread_t h2;

    pthread_create(&h2, NULL, (void *)hilo2, NULL);
    pthread_create(&h1, NULL, (void *)hilo1, NULL);
    pthread_join(h2, NULL);
    printf("Fin\n");
    exit(1);
}

void hilo1(void) {
    int veces = 1;
    while(veces <= 10) {
        printf("HOLA ", veces);
    }
}

void hilo2(void) {
    int veces = 10;
    while(veces != 0) {
        printf("Mundo \n");
    }
}

```

2

pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER; INICIALIZAR

pthread_mutex_t m2 = " " " " " " ;

Agrego Mutex →

Punteros →

CREAR HILO

Espero terminación de hilo2 (h2).

NO ALMACENO EL ESTADO DE SALIDA DEL HILO2.

1, 2) ...

A(M)

w(A)

se

s(sa)

B(M)

w(s)

se

s(sa)

P.HOLA → **P.MUNDO**

3.- Realizar el enunciado del siguiente programa (2,5)

```

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
void f_signal(int);
int cont = 0;
int main(void) {
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigprocmask(SIG_BLOCK, &set, NULL);
    printf("proceso %d\n", getpid());
    signal(SIGALRM, f_signal);
    signal(SIGINT, f_signal);
    alarm(1);
    while(cont != 10);
    sigprocmask(SIG_UNBLOCK, &set, NULL);
    while(1);
    exit(0);
}

void f_signal(int s) {
    if (s == 14) { printf("SIGALRM\n"); cont++; alarm(1); }
    if (s == 2) { printf("----SIGINT\n"); signal(SIGINT, SIG_DFL); }
}

```

Creación de una variable set (ordenación de señales que se pueden bloquear de bloquear o manipular).

Agrega la función SIGINT a Set

Bloqueo (SIG_BLOCK)

las señales de set.

indica señales bloqueadas anteriormente

3 X

4.- Obtener el tiempo medio de espera y el porcentaje de sobrecarga del sistema operativo para la siguiente definición de procesos para un algoritmo Round Robin con un quantum de 10ms. Interrupciones no enmascarables. Orden de Listos PA y PB. Indicar con "E" ejecución, con "D" disco, con "C" cinta y con "S" cuando el disco es necesario (2,5)


NO LO VIMOS

```

Void M101 (void) {
    int veces = 1
    while (veces <= 10) {
        pthread_mutex_lock (&m1);
        printf ("M101", veces);
        veces++;
        pthread_mutex_unlock (&m1);
    }
}

Void M102 (void) {
    int veces = 10
    while (veces != 0) {
        pthread_mutex_lock (&m2);
        printf ("Mundo Vn");
        veces--;
        pthread_mutex_unlock (&m2);
    }
}

```



Sistemas Operativos
Primer Examen Parcial 13/07/15

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

Apellido y Nombre:

PA	CPU 25 ms., E/S disco 20ms, CPU 15 ms, E/S cinta 20 ms., CPU 5ms.
PB	CPU 15 ms., E/S disco 20 ms., CPU 25 ms.

⑤ Un proceso es un programa en ejecución
 Instancia de un programa ejecutado en un computador
 Entidad que se le asigna a un CPU y está la ejecuta
 Entidad que presenta un estado actual, una serie de instrucciones y recursos del sistema asociados al proceso

⑥ Un proceso zombie es un proceso que no se encuentra en ejecución ya que terminó de manera normal o anormal
 La información del proceso queda en el sistema
 El proceso zombie se da cuando el hijo termina antes de una llamada `waitpid()`
 Cuando el padre recupera la información de finalización del proceso, este dejará de ser zombie.

⑦ Un SMP tiene una memoria fuertemente acoplada → Varios procesadores comparten un espacio de memoria común.
 Compartida por CPUs que se encuentran sobre el mismo bus
 En cambio, un cluster tiene memoria débilmente acoplada ya que se representa n computadores distintos con una o más (

⑧ F0FS no es viable en un sist. uniprosesor porque si se enciende un proceso largo y otro corto, el último tardará de

Si aplicamos **SMP** ya que tendremos una **única cola de procesos** 'plegar' => **A mayor número de CPU's m**

ARQUITECTURA MULTISERVIDOR



Sistemas Operativos
Primer Examen Parcial 13/07/15

UTN FRD

Ingeniería en Sistemas de Información

Dpto. Ingeniería en Sistemas de Información

Área: Computación

Apellido y Nombre:.....

SISTEMAS OPERATIVOS - RESPUESTAS


1.a) Complete la línea punteada en el código para que se cumpla el objetivo.

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

int tarea(int);

int main(void) {
    pid_t pid , pidTer;
    int status1 = 0,status2 = 0,status3 = 0;
    pid = fork();
    if (pid == 0) {
        pid = fork();
        if (pid > 0 ) {
            pid = fork() ;
            if (pid == 0 )
                exit(tarea(getpid()));
            else {
                pidTer = wait(&status1);
                printf("termino %d informa %d\n",pidTer,status1/256);
                pidTer = wait(&status2);
                printf("termino %d informa %d\n",pidTer,status2/256);
                exit(status1 && status2);
            }
        }
        else
            exit(tarea(getpid()));
    } else{
        if (pid > 0 ) {
            pidTer = wait(&status3);
            printf("termino %d informa %d\n",pidTer,status3/256);
        }
        exit(pid);
    }

    int tarea(int num) {
        printf("tarea %d\n",num);
        return num / 2000 ;
    }
}
```



Sistemas Operativos
Primer Examen Parcial 13/07/15

UTN FRD

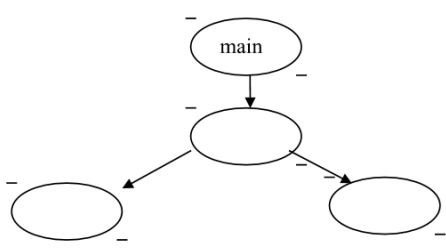
Ingeniería en Sistemas de Información

Dpto. Ingeniería en Sistemas de Información

Área: Computación

Apellido y Nombre:.....

1.b) Grafique la estructura jerárquica de procesos que se creó.



2.- Completar el siguiente programa

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


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

pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER ;

int main(void) {
    pthread_t h1 ;
    pthread_t h2 ;
    pthread_mutex_lock(&m1) ;

    pthread_create(&h2,NULL,(void *)Hilo2,NULL);
    pthread_create(&h1,NULL,(void *)Hilo1,NULL);
    pthread_join(h2,NULL);
    printf("Fin\n");
    exit(1);
}

void Hilo1(void) {
    int veces = 1 ;
    while(veces <= 10 ) {
        pthread_mutex_lock(&m2) ;
        printf("%d) Hola ",veces);
        veces ++ ;
        pthread_mutex_unlock(&m1) ;
    }
}
```



Sistemas Operativos
Primer Examen Parcial 13/07/15

UTN FRD

Ingeniería en Sistemas de Información

Dpto. Ingeniería en Sistemas de Información

Área: Computación

Apellido y Nombre:.....

```
int veces = 10 ;
while(veces != 0 ) {
    pthread_mutex_lock(&m1) ;
    printf("Mundo \n");
    veces -- ;
    pthread_mutex_unlock(&m2) ;
}
}
```

3.

Realice un programa que muestre el pid del proceso, bloquee la señal SIGINT, luego que redefina el handle de la señal SIGALRM y SIGINT. Por un lapso de 10 segundos, cada 1 segundo muestre por pantalla la leyenda "SIGALARM" y que la segunda vez que el proceso reciba la señal SIGINT termina el proceso (después de pasados 10 segundos). Al presionarse SIGINT (luego de pasados 10 segundos) imprime en pantalla la leyenda "SIGINT".

4.

5 Enumere 4 oraciones que definan qué es un proceso (2,5)

Programa en ejecución.

Instancia de un programa ejecutado en un computador.

"Espíritu animado" de un programa.

Entidad asignada a una CPU y ejecutada por ésta.

Unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual y un conjunto de recursos del sistema asociados.

6.- En un entorno Unix, ¿Qué es un proceso zombie? (2,5)

Es un proceso que ya no esta en ejecución, ya sea por una terminación normal o anormal o porque fue abortado; pero aún queda su información registrada en el sistema para que pueda consultarla el padre.

Obsérvese el código del shell, cuando el proceso padre hace un fork() (crea un hijo), y luego hace una llamada a la función waitpid() (si el hijo termina antes de la llamada a waitpid() por parte del padre, el hijo pasa a estado zombie) y luego el padre necesita que no "desaparezca su rastro" para recuperar el status de finalización del proceso hijo. Una vez que el padre ha recuperado la información de finalización del proceso hijo a través de waitpid(), el hijo deja de estar en estado zombie y pasa al estado terminado [Tanenbaum, Sistemas Operativos Modernos, 3ra. ed., Pág. 744].

7.- ¿En qué se diferencia un SMP de un Cluster? (2,5)

En cuanto a la memoria. Un SMP tiene una memoria fuertemente acoplada, es compartida por n CPU's que se encuentran sobre el mismo bus, en la misma motherboard, en un mismo computador Fig. 4.9, Pag.175. Mientras que un Cluster tiene una memoria débilmente acoplada o distribuida, puesto que se trata de n computadores distintos c-u con una o más CPU's y con sus propias memorias.

8.- ¿Cuáles serían las razones por cuales decimos que la política FCFS no es viable en un sistema uniprocesador, pero sin embargo, podría aplicarse en un SMP? (2,5)

FCFS no es viable en un sistema uniprocesador porque si se encola un proceso largo y otro corto, éste último deberá esperar demasiado tiempo para su ejecución. Sin embargo, podría aplicarse en un SMP tradicional, en donde por lo general, hay una única cola de procesos a ejecutar (arquitectura multiservidor) y habiendo n cpu's, el proceso más corto tendría una menor probabilidad de sufrir inanición. Esto esta respaldado por los trabajos de Saver C et. al. : a mayor número de cpu's menos importante es la sofisticación del algoritmo de planificación.

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

① Proceso ⇒ Crea un qto de procesos
Asigna a c/uho una tarea.
c/ proceso debe indicar a su proceso creador
ESTADO 1 ⇒ TAREA RESUELTA ESTADO 0 ⇒ TAREA NO RESUELTA