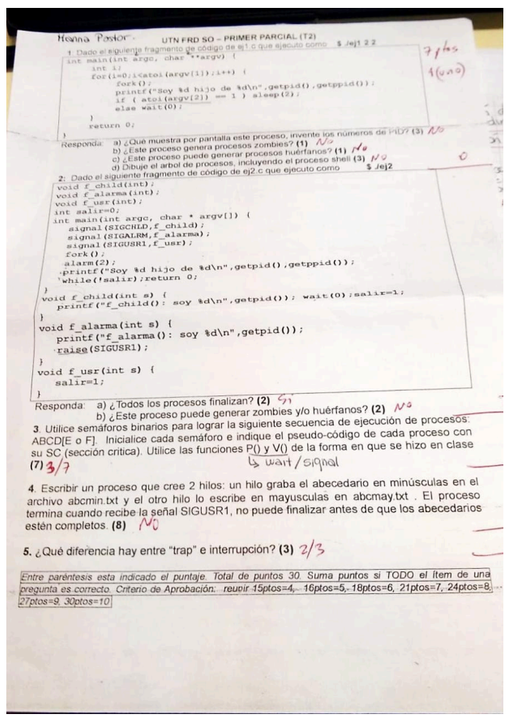
Parcial 12/05 T2

lunes, junio 05, 2023 2:36 PM



Nombre Padrón UTN FRD SO – PRIMER PARCIAL (T2)

1: Dado el siguiente fragmento de código de ej1.c que ejecuto como $ ./ej1 2 2

```
int main(int argc, char **argv) {
    int i;
    for(i=0;i<atoi(argv[1]);i++) {
        fork();
        printf("Soy %d hijo de %d\n",getpid(),getppid());
        if ( atoi(argv[2]) == 1 ) sleep(2);
        else wait(0);
    }
    return 0;
}
```

Responda: a) ¿Qué muestra por pantalla este proceso, invente los números de PID? (3)
b) ¿Este proceso genera procesos zombies? (1)
c) ¿Este proceso puede generar procesos huérfanos? (1)
d) Dibuje el arbol de procesos, incluyendo el proceso shell (3)

2: Dado el siguiente fragmento de código de ej2.c que ejecuto como $ ./ej2

```
void f_child(int);
void f_alarma(int);
void f_usr(int);
int salir=0;
int main(int argc, char * argv[]) {
  signal(SIGCHLD,f_child);
  signal(SIGALRM,f_alarma);
  signal(SIGUSR1,f_usr);
  fork();
  alarm(2);
  printf("Soy %d hijo de %d\n",getpid(),getppid());
  while(!salir);return 0;
}
void f_child(int s) {
  printf("f_child(): soy %d\n",getpid()); wait(0);salir=1;
}
void f_alarma(int s) {
  printf("f_alarma(): soy %d\n",getpid());
  raise(SIGUSR1);
}
void f_usr(int s) {
  salir=1;
}
```

Responda: a) ¿Todos los procesos finalizan? (2)
b) ¿Este proceso puede generar zombies y/o huérfanos? (2)

3. Utilice semáforos binarios para lograr la siguiente secuencia de ejecución de procesos: ABCD[E o F]. Inicialice cada semáforo e indique el pseudo-código de cada proceso con su SC (sección crítica). Utilice las funciones P() y V() de la forma en que se hizo en clase (7)

4. Escribir un proceso que cree 2 hilos: un hilo graba el abecedario en minúsculas en el archivo abcmin.txt y el otro hilo lo escribe en mayusculas en abcmay.txt . El proceso termina cuando recibe la señal SIGUSR1, no puede finalizar antes de que los abecedarios estén completos. (8)

5. ¿Qué diferencia hay entre "trap" e interrupción? (3)

Entre paréntesis esta indicado el puntaje. Total de puntos 30. Suma puntos si TODO el ítem de una pregunta es correcto. Criterio de Aprobación: reunir 15ptos=4, 16ptos=5, 18ptos=6, 21ptos=7, 24ptos=8, 27ptos=9, 30ptos=10

```
Int main(int argc, char**argv){

        int i;
       For(i=0; i<atoi(argv[1]), i+++){
              Fork();
              Printf("soy %d hijo de %d\n", getpid(), getppid());
              If (atoi(argv[2])==1){
                     Sleep(2);
              }else {
                     Wait(0);
              }
       }
       Return 0;
}
```

① $.ej1 2 2 → parametros

char **argv → Array de punteros

→ sirve para pasar datos al programa principal.

→ Cuando el usuario ingresa un dato

argc = 2

argv = 2

```
for( i=0, i<atoi(argv[1]), i++) {
     fork(),
     printf("soy %d hijo de %d\n", getpid(), getppid()),
  if (2==1)
     sleep(2),
  else
     wait(0),
 }
  return 0,
}
```

i<2

A) ¿Qué muestra por pantalla este proceso?

El proceso muestra en pantalla:

Soy 1807 (pid hijo) hijo de 1806 (pid padre) (main)
Soy 1808 hijo de 1807

Soy 1807 (pid hijo) hijo de 1806 (pid padre) (main)
Soy 1810 hijo de 1807

Soy 1808 hijo de 1807
Soy 1809 hijo de 1808

COMO EL PRINTF SE ENCUENTRA DEBAJO DEL FORK LO HEREDA EL HIJO Y EL PADRE TAMBIÉN LO REALIZA.
Por lo tanto siempre el proceso padre y el hijo hacen el printf

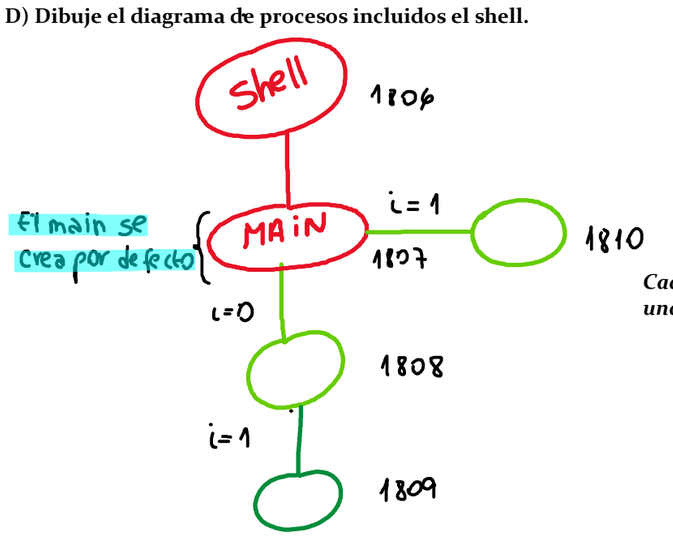
B) ¿Este proceso genera procesos zombies?

El proceso no genera procesos zombies ya que en el wait(0) se retorna la información del proceso hijo al padre.

C) ¿Este proceso genera procesos huerfanos?

El proceso no genera procesos huerfanos ya que el padre espera mediante el wait(0) la finalización del proceso hijo.

D) Dibuje el diagrama de procesos incluidos el shell.



Cada vez que se realiza un fork, el hijo copia una imagen del padre (código y variables)´

② El siguiente programa se ejecuta como $./ej2

```
int main(int argc, char * argv[])
```

argc → 0
argv → ∄ (VACIO)

Flujo del código: Se declaran las funciones f_child, f_alarma, usuario (usr), y se le asigna la función correspondiente.
Luego se realiza un fork(); tanto el padre como el hijo presentan la función alarma(2)
y una alarma establecida en 2 segundos.
Cuando esta se activa llama a la función void alarm tanto el padre como el hijo imprimen el printf
Soy 2002
Soy 2001

Se llama a la función usuario que modifica el valor de la variable salir de cero a uno en ambos
procesos.
Luego de esto se imprime:
se imprime soy 2002 proceso hijo creado 2001
se imprime soy 2001 proceso hijo creado por 2000
Por último como en el ciclo el valor de la variable salir es distinto de cero retorna cero y finaliza el programa.

A) ¿Todos los procesos finalizan?

Esto depende de la prioridad que realice el sistema operativo ya que

1. Si le da prioridad al proceso padre el padre termina y el hijo finaliza pero no se retorna la información al padre.
2. En caso contrario si se le da prioridad al hijo este finaliza, se llama a la función SIGCHLD y se retorna la información del proceso hijo al proceso padre y este último finaliza.

B) ¿ Este proceso puede generar procesos huerfanos y/o zombies?

En el caso 1 el proceso hijo sería huerfano porque el padre termina antes que el proceso hijo y zombie ya que no se retorna información al init cuando este queda huerfano.
En el caso 2 el proceso no será huerfano ni zombie ya que se llama a la función SIGCHLD y esta contiene el wait(0) que genera la espera y el retorno de la información del proceso hijo.

3

0 → Uno semáforos!

Utilice semáforos binarios para lograr ABCD[E o F]

A	B	C	D	E	F
Wait(Sa)	Wait(Sb)	Wait(Sc)	Wait(Sd)	Wait(Sef)	Wait(Sef)
SC	SC	SC	SC	SC	SC
Signal(Sb)	Signal(Sc)	Signal(Sd)	Signal(Sef)	Signal(Sa)	Signal(Sa)

Inicial ⇒ 1 0 0 0 0

4

```
Void Hilo1 (void);
Void Hilo2 (void);
char letramin = 'a';
char letramay = 'A';

Int main(void){

Signal(sigusr1, h_sigusr1);
File * fd1 = fopen("abcmin.txt", "w");
File * fd2 = fopen("abcmay.txt", "w");

int salir = 0;

Pthread &h1;
Pthread &h2;

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

pthread_join(&h1, NULL);
pthread_join(&h2, NULL);

While(!salir){

        If(letramin !='z' && letramay != 'Z'){
                pause();
        } else {
                raise(SIGUSR1);
                }

        }

        fclose(fd1);
        fclose(fd2);
        Return 0;
}

Void HIlo1(void){

        While(letramin <= 'z'){
        fprintf(fd1, "%c\n", letramin);
        Letramin ++;
        }
}

Void Hilo2(void){

        While(letramay <= 'Z'){
         fprintf(fd2, "%c\n", letramay);
        letramay ++;
        }
}

Void  h_sigusr1(int s){
        Salir = 1;
}
```

5 Diferencia trap e interrupción

Un **trap** es una forma de interrupción pero que se origina dentro del propio proceso interrumpido, asociados una condición de error o excepción.

Una **interrupción** es algo externo al proceso en ejecución, por ejemplo puede ser que se haya agotado el tiempo asignado para la ejecución del mismo entonces se produce una interrupción de reloj.