Examen SSOO Práctico Resuelto-Enero 2021-2022

Ejercicio1.c

Nos piden crear dos procesos hijos en paralelo, uno que abra la calculadora y otro el reloj. Los nombres de los ejecutables son pasados por línea de comandos.

Ejemplo de ejecución: ./a.out gnome-calculator xclock;

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

void calculadora(char *f){
if (execlp(f,f,(char*)NULL)<0)//ERROR EN EXCECLP
{
perror("exec");
printf("ERRNO= %d\n",errno );
exit(EXIT_FAILURE);
}
}

void reloj(char *f){
if (execlp(f,f,(char*)NULL)<0)//ERROR EN EXCECLP
{
perror("exec");
printf("ERRNO= %d\n",errno );
```

```
exit(EXIT_FAILURE);
}
}

int main(int argc, char  *argv[])
{ int n,status;
pid_t pid, hijo_pid;

if (argc!=3)
{
printf("Error en la linea de argumentos-->Ejecutable+gnome-calculator+xclock\n");
}

for (int i = 0; i < 2; ++i)//2hijos
{
pid=fork();
if (pid==0)
{
if (i==0)
{printf("Proceso Hijo,mi pid es: %d, mi padre tiene el pid: %d y voy a abrir la calculadora \n",getpid(),getppid());
calculadora(argv[1]);//Abrir calculadora
}
else{
printf("Proceso Hijo,mi pid es: %d, mi padre tiene el pid: %d y voy a abrir el reloj \n",getpid(),getppid());
reloj(argv[2]);//Abrir reloj
}
}
else if (pid>0)
```

```
{
//En el examen el proceso padre no imprimia nada
}
else{
printf("ERROR EN LA LLAMADA FORK\n");
printf("ERRNO: %d\n",errno );
exit(EXIT_FAILURE);

}
}

while ( (hijo_pid=wait(&status)) > 0 )
{
if (WIFEXITED(status))
{
printf("Proceso padre %d, hijo con PID %ld finalizado, status = %d\n", getpid(), (long int)hijo_pid, WEXITSTATUS(status));
}
else if (WIFSIGNALED(status)) //Para seniales como las de finalizar o matar
{
printf("Proceso padre %d, hijo con PID %ld finalizado al recibir la señal %d\n", getpid(), (long int)hijo_pid, WTERMSIG(status));
}
}
if (hijo_pid==(pid_t)-1 && errno==ECHILD) //Entra cuando vuelve al while y no hay más hijos que esperar
{
printf("Proceso padre %d, no hay mas hijos que esperar. Valor de errno = %d, definido como: %s\n", getpid(), errno, strerror(errno));
}
else
```

{

printf("Error en la invocacion de wait o waitpid. Valor de errno = %d, definido como: %s\n", errno, strerror(errno));

exit(EXIT_FAILURE);

}

return 0;

}

Ejercicio2.c

Nos piden crear dos hebras, una “escritor” y otra “lector”, en el que ambos pueden acceder a la variable compartida llamada “variable” inicializada inicialmente a -1.

Ambas no pueden ni escribir y leer a la vez, pudiéndose la intercalación entre uno y otro.

Cada hebra se ejecutará un numero de iteraciones pasados por línea de argumentos

Ejemplo de invocación: ./a.out 3

Nota: - Casi siempre sale, por ejemplo, en el ejemplo de arriba podría salir 3 lectores-3 escritores o viceversa porque son procesos muy cortos pero también puede suceder este caso:

```
-->Escritor 140552939398912, escribo el numero 6
<--Lector140552931006208, leo el numero 6
<--Lector140552931006208, leo el numero 6
<--Lector140552931006208, leo el numero 6
-->Escritor 140552939398912, escribo el numero 6
-->Escritor 140552939398912, escribo el numero 9
```

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <pthread.h>

#include <errno.h>

#include <unistd.h>


int variable=-1;

pthread_mutex_t mutex= PTHREAD_MUTEX_INITIALIZER;//Iniciar mutex
```

```
void*Escritor(void*t){

int s,cantidad;
int*x;
x=(int*)t;

for (int i = 0; i < (*x); ++i)
{
pthread_mutex_lock(&mutex);

cantidad=rand()%10+1;
variable=cantidad;
printf("-->Escritor %ld, escribo el numero %d\n",pthread_self(),variable );
pthread_mutex_unlock(&mutex);

}
pthread_exit(NULL);
}

void*Lector(void*t){

int s,cantidad;
int*x;
x=(int*)t;
for (int i = 0; i < (*x); ++i)
{
pthread_mutex_lock(&mutex);
```

```
printf("<--Lector%ld, leo el numero %d\n",pthread_self(),variable );
pthread_mutex_unlock(&mutex);

}

pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
srand(time(NULL));//Semilla numeros aleatorios
int n_iteraciones;

if (argc!=2)
{
printf("Error en la linea de comandos\n");
printf("./exe n_iteraciones\n");
exit(EXIT_FAILURE);
}
n_iteraciones=atoi(argv[1]);
pthread_t hilo[2];

for (int i = 0; i < 2; ++i)
{
if (i==0)
{
if (pthread_create(&hilo[i],NULL,(void*)Escritor,(void*)&n_iteraciones))
{
printf("Error a la de crear un hilo\n");
```

```
exit(EXIT_FAILURE);
}
}
else{
if (pthread_create(&hilo[i],NULL,(void*)Lector,(void*)&n_iteraciones))
{
printf("Error a la de crear un hilo\n");
exit(EXIT_FAILURE);
}
}
}
for (int i = 0; i < 2; ++i)
{

if (pthread_join(hilo[i],NULL))
{
printf("Error en la espera de la hebra-->Codigo de error: %d\n",errno );
exit(EXIT_FAILURE);
}




}
return 0;
}
```