Examen Enero 2020 Corregido.pdf

amgoal

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Politécnica Superior de Córdoba
Universidad de Córdoba

Accede al documento original

EXAMEN DE SISTEMAS OPERATIVOS ENERO 2020

Ejercicio 1

Realice un programa que cree dos hijos.

- El primero calculará el factorial del número pasado como argumento. Deberá de mostrar por pantalla antes de terminar el siguiente mensaje: "Soy el hijo PID , mi padre PPID el factorial de A es X"
- El segundo abrirá la calculadora y mostrará el mensaje: "Soy el hijo PID , mi padre PPID y voy a abrir la calculadora."
- El padre después de crear los hijos deben de esperar a los hijos y debe mostrar el estado de finalización de la siguiente manera: "Padre PID, ha finalizado el hijo PID con el estado ESTADO" por cada hijo. Mostrando las señales de ERRNO

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

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

    if(argc != 2){

        printf("\nNumero de argumentos incorrecto\n");
        exit(EXIT_FAILURE);
    }

    pid_t wpid;
    pid_t fact = fork();
    int status;

    if(fact < 0){

        printf("\nFactorial creado incorrectamente\n");
        exit(EXIT_FAILURE);
    }

    else if(fact == 0){

        printf("\nEntrando en factorial\n");

        execl("./factorial", "./factorial", argv[1], NULL);

        printf("\nFactorial abierto incorrectamente\n");
        exit(EXIT_FAILURE);
    }

    pid_t calc = fork();

    if(calc < 0){

        printf("\nCalculadora creado incorrectamente\n");
        exit(EXIT_FAILURE);
    }
```

```
    else if(calc == 0){

        printf("\nSoy el hijo %d , mi padre %d y voy a abrir la calculadora.\n", getpid(), getppid());
        execlp("gnome-calculator", "gnome-calculator", NULL);

        printf("\nCalculadora abierto incorrectamente\n");
        exit(EXIT_FAILURE);
    }

    while((wpid=waitpid(-1, &status, WUNTRACED | WCONTINUED)) > 0){

        if(WIFEXITED(status)){
            printf("\nPadre %d, ha finalizado el hijo %d con el estado %d", getpid(), wpid,
WEXITSTATUS(status));
        }

        if(WIFSIGNALED(status)){
            printf("\nPadre %d, ha finalizado el hijo %d con el estado %d", getpid(), wpid,
WTERMSIG(status));
        }

        if(WIFSTOPPED(status)){
            printf("\nPadre %d, ha finalizado el hijo %d con el estado %d", getpid(), wpid,
WSTOPSIG(status));
        }
    }
    exit(EXIT_SUCCESS);
}
```

Factorial ejercicio 1

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

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

    if(argc!=2){
        fprintf(stderr, "\nNumero de argumentos incorrecto: ./factorial <Numero a calcular>\n");
        exit(EXIT_FAILURE);
    }

    int fact=1;

    for(size_t i = 1; i <= atoi(argv[1]); i++){
        fact *= i;
    }

    printf("\nSoy el hijo %d , mi padre %d el factorial de %d es %d\n", getpid(), getppid(), atoi(argv[1]),
fact);
    exit(EXIT_SUCCESS);
}
```

Ejercicio 2

Realice un programa que cree un número N de hilos, donde N será un argumeto del programa principal. Se tendrán dos variables globales "par" e "impar" . A cada hilo se le pasará su índice (1,...N).

- Cada hilo genera 100 valores aleatorios entre 0 y 10 que se añadirán a la varable par o impar dependiendo del índice del hilo. (Variable par si el índice es par . Variable impar si el índice es impar). El hilo devolverá la suma total de los número generados. Antes de finalizar el hilo duerme durante 10 nanosegundos.
- El padre comprobará que la suma devuelta por los hilos sea igual a la suma de las variables par e impar.

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <pthread.h>
#include <time.h>

//Inicializamos las variables globales y los mutex que protegeran a los mismos
int par = 0;
int impar = 0;
pthread_mutex_t mpar = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mimpar = PTHREAD_MUTEX_INITIALIZER;

//Creamos la estructura que llevaran los hilos (indice y suma)
struct Hilo{

    int indice;
    int suma;
};

//Funcion de los hilos
void *func(void * arg){

    //Creamos la estructura del hilo e inicializamos la suma a 0
    struct Hilo *hilo = (struct Hilo*) arg;
    hilo->suma = 0;

    //Generamos 100 numeros del 0 al 10
    for(size_t i = 0; i < 100; i++){

        int num = (int)rand() % 11;

        //Si es par se suma a par
        if((num % 2) == 0){

            pthread_mutex_lock(&mpar);
            par += num;
            pthread_mutex_unlock(&mpar);
        }

        //Si es impar a impar
        else{

            pthread_mutex_lock(&mimpar);
            impar += num;
```


```
            pthread_mutex_unlock(&mimpar);
        }

        //SUmamos en suma todos los valores
        hilo->suma += num;
    }

    //Imprimimos el indice y la suma antes de terminar el hilo
    printf("\nHilo %d= %d", hilo->indice, hilo->suma);
    pthread_exit((void*) hilo);
}

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

    if(argc != 2){

        fprintf(stderr, "\nNumero de argumentos incorrecto\n");
        exit(EXIT_FAILURE);
    }

    //Creamos un vector de estructuras y uno de hilos del tamañado pasado por argumentos
    int tam = atoi(argv[1]);
    srand(time(NULL));
    struct Hilo *estruc[tam];
    pthread_t hilos[tam];

    //Para cada hilo
    for(size_t i = 0; i < tam; i++){

        //Reservamos la memoria
        estruc[i] = malloc(sizeof(struct Hilo));

        //Verificamos que este correctamente reservada
        if(estruc[i] == NULL){

            fprintf(stderr, "\nMemoria reservada incorrectamente\n");
            exit(EXIT_FAILURE);
        }

        //Añadimos el indice
        estruc[i]->indice = (i+1);

        //Creamos el hilo
        if(pthread_create(&hilos[i], NULL, &func, estruc[i]) != 0){

            fprintf(stderr, "\nHilo llamado incorrectamente\n");
            exit(EXIT_FAILURE);
        }
    }

    //Definimos la suma total de los hilos
    int total = 0;
```

```
    //Para cada hilo
    for(size_t i = 0; i < tam; i++){

        struct Hilo *ret;

        //Recogemos los valores
        if((pthread_join(hilos[i], (void**) &ret) != 0)){

            fprintf(stderr, "\nHilo recogido incorrectamente\n");
            exit(EXIT_FAILURE);
        }

        //Se añade a total la suma de cada hilo
        total += ret->suma;
        printf("\nPADRE -> Recogí el hilo %d con suma= %d", ret->indice, ret->suma);
        free(ret);
    }

    //Imprimimos valores
    printf("\nLa suma de los hilos es %d\nLa suma de pares (%d) e impares (%d) es %d", total, par, impar,
(par + impar));
    pthread_mutex_destroy(&mpar);
    pthread_mutex_destroy(&mimpar);
}
```