Sistemas Operativos y Redes (E0224)
Año 2021

Trabajo Práctico N°3

Grupo N°4:

Ignacio Hamann - 68410/3
Juan Pablo Elisei - 68380/5
Tomás Tavella - 68371/4

Resumen

Para este trabajo se desarrolla la implementación en C de un *buffer ping-pong* utilizando memoria compartida, semáforos y colas de mensajes; como solución de un problema de productor-consumidor.

Facultad de Ingeniería
Universidad Nacional de La Plata

Índice

1. Enunciado

Se desea escribir un programa que genere un vector de N elementos con los valores de 1 a N almacenados como double. El valor de N se pasará en la línea de comando en el momento de iniciar el programa.

Si el programa recibe la señal `SIGUSR1` deberá crear un proceso hijo, que imprimirá los valores almacenados en el vector separados por comas y los imprimirá en pantalla, junto con la identificación del proceso y del proceso padre.

Si el programa recibe la señal `SIGUSR2` deberá crear un hilo, este hilo deberá modificar cada elemento del vector, multiplicándolo por si mismo, imprimir en pantalla los valores obtenidos separados por espacios y devolver el valor de la suma de los elementos del vector cuando termine.

El programa deberá llevar la cuenta de la cantidad de hijos e hilos generados, imprimir los resultados devueltos por cada hilo, verificar que N no sea mayor que 20 y mostrar el estado con el que termina cada hijo (PID y Estado). Una vez que se generen 10 hilos, no deberá generar más hilos cuando reciba la señal `SIGUSR2` e imprimirá un cartel avisando que la cantidad máxima de hilos ha sido alcanzada.

Si el programa recibe la señal `SIGTERM`, esperará que se completen los hilos e hijos pendientes si los hubiera y terminará.

Tener en cuenta las siguientes indicaciones:

- Cuidar que no queden procesos en estado *zombie*.
- No utilizar variables globales dentro de los hilos.
- El espacio de memoria para el vector con los N elementos debe reservarse dinámicamente mediante la función `malloc()`.

2. Interpretación del problema

El programa debe recibir un entero como argumento por línea de comandos, debe crear un vector de esa cantidad de elementos de tipo *double* con la función `malloc()` para reservar la memoria, debe preparar los *signal handlers* necesarios y quedar en espera de las señales.

Si la señal es `SIGUSR1`, el handler debe hacer un fork y el hijo resultante debe imprimir el contenido del vector separado con comas, su PID, y el PID de su padre.

Si la señal es `SIGUSR2`, se debe crear un *thread* o hilo que multiplique a los elementos por si mismos e imprima los valores separados por espacio, y al finalizar devuelva la suma de los elementos. No puede haber más de 10 hilos en total.

Además, el proceso padre tiene que imprimir el estado en el cual terminaron los hijos, verificar que no quede ninguno en estado *zombie*, y avisar si se llega al máximo de hilos. Si se recibe la señal `SIGTERM`, se espera a que termine todos los hijos e hilos y se termina el programa.

3. Resolución

Se utilizaron las siguientes bibliotecas de C para poder llevar a cabo la resolución del problema planteado:

- Para el manejo de procesos se utilizaron:
  - `<sys/types.h>`
  - `<sys/wait.h>`
- `<signal.h>` para el manejo de señales.

- `<pthread.h>` para el manejo de *threads* o hilos.

En el `main()` del programa, se utiliza la función `malloc()` (definida en las bibliotecas estándar) para reservar el espacio de memoria para el vector de N elementos.

Para el manejo de señales, se utiliza la función `signal()` (parte de `<signal.h>`), que se encarga de instalar *signal handlers* para las distintas señales que el programa va a recibir. Esta función toma dos parámetros: el número o nombre de la señal a manejar; y la función que se va a encargar del manejo de la señal.

Para el manejo de procesos padres e hijos, fueron de utilidad las siguientes funciones (parte de las bibliotecas `<unistd.h>` y `<sys/wait.h>`):

- `fork()`: Genera una copia de la imagen del proceso padre y se la asigna al proceso hijo, al cual se le asignan nuevos identificadores. Para distinguir al padre del hijo, esta función devuelve 0 al ser llamada en el hijo y el PID del hijo al ser llamada en el padre.
- `getpid()`: Devuelve el PID del proceso que hace la llamada al sistema.
- `getppid()`: Devuelve el PID del proceso padre del proceso que hace la llamada al sistema.
- `waitpid()`: Suspende la ejecución del proceso que llama a la función hasta que el hijo con el PID especificado termine.
- `wait()`: Suspende la ejecución del proceso que llama a la función hasta que algún proceso hijo termine.

En tanto al manejo de los *threads*, se utilizaron las siguientes funciones:

- `pthread_create()`: Crea un nuevo *thread*, le asigna la función que este va a ejecutar y le pasa los argumentos.
- `pthread_exit()`: Finaliza el *thread* en el que se llama, y devuelve un valor que se le especifica a la función que creó el *thread*.
- `pthread_join()`: Detiene el thread en ejecución hasta que otro hilo termina, y recupera el valor que este devolvió.

Para compilar el código a un archivo binario ejecutable, se debe utilizar el comando `gcc` (*GNU C Compiler*) en la terminal, agregando el argumento `-pthread` para permitir la utilización de *threads*:

```
$ gcc entregable2.c -o vector_procesos_hilos -pthread
```

Donde `vector_procesos_hilos` es el archivo compilado ejecutable. Para correrlo, se debe llamar al archivo seguido del número de elementos que se quiera el vector:

```
$ ./vector_procesos_hilos <N>
```

Si no se pasa la cantidad de argumentos requerida, el programa sugiere un ejemplo con la sintaxis correcta.

Para el envío de señales al programa, se debe utilizar el comando `kill`. Una forma más amigable con el usuario para no tener que recordar el número de la señal ni el PID del proceso es utilizar los siguientes argumentos:

```
$ kill -s <nombre_señal> $(pidof vector_procesos_hilos)
```

Donde `nombre_señal` se corresponde con `SIGUSR1`, `SIGUSR2` o `SIGTERM`, y el comando `pidof` devuelve el PID del programa en ejecución sin necesidad de buscarlo manualmente.

3.1. Pseudocódigo

A continuación se muestra un pseudocódigo correspondiente con el código en C.

```
INICIO
    Declaracion y asignacion de variables y funciones
    if(Cantidad de argumentos es menor a 2){
        Imprime error en pantalla;
        salir;
    }

    Se convierte la cantidad de elementos de cadena de caracteres a entero;
    if(Cantidad cantidad de elementos es mayor a 20){
        Imprime error en pantalla;
        salir;
    }

    Imprime en pantalla el PID del proceso;
    Reserva el espacio para el vector de elementos de tipo double y los genera;
    Se asignan las signals a sus respectivos handlers
    Se imprime en pantalla que se esta a la espera de las signals y se queda esperando;
FIN

trapUSR1(){
    if(La cantidad de hijos es menor al maximo){
        if(Se crea un proceso y el proceso actual es un hijo){
            Se muestran los elementos del vector separados por comas junto con la
     identificacion del proceso padre;
        }
        else{
            Se incrementa el contador de hijos;
        }
    else{
        Imprime que se llego al maximo de hijos;
    }
}

trapUSR2(){
    Se inicializan variables;
    if(La cantidad de hilos es menor al maximo){
        Se crea un hilo nuevo, llamando a thread_function;
        Se recupera el valor de suma que devuelve thread_function;
        Se imprime la suma en pantalla;
    }
    else{
        Imprime que se llego al maximo de hilos;
    }
}

trapTERM(){
    Se espera a que terminen todos los hijos e hilos;
    Se liberan los recursos de los hijos una vez que terminan;
}

thread_function(){
    Se multiplican e imprimen los elementos del vector por si mismos, separados por espacios;
    Se realiza la suma de todos los elementos;
    Se devuelve la suma;
}
```

```
%%
%% This is file `.tex',
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
%% fileerr.dtx  (with options: `return')
%%
%% This is a generated file.
%%
%% The source is maintained by the LaTeX Project team and bug
%% reports for it can be opened at https://latex-project.org/bugs/
%% (but please observe conditions on bug reports sent to that address!)
%%
%%
%% Copyright (C) 1993-2020
%% The LaTeX3 Project and any individual authors listed elsewhere
%% in this file.
%%
%% This file was generated from file(s) of the Standard LaTeX `Tools Bundle'.
%% -----------------------------------------------------------------------
%%
%% It may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either version 1.3c
%% of this license or (at your option) any later version.
%% The latest version of this license is in
%%    https://www.latex-project.org/lppl.txt
%% and version 1.3c or later is part of all distributions of LaTeX
%% version 2005/12/01 or later.
```

```
%%
%% This file may only be distributed together with a copy of the LaTeX
%% 'Tools Bundle'. You may however distribute the LaTeX 'Tools Bundle'
%% without such generated files.
%%
%% The list of all files belonging to the LaTeX 'Tools Bundle' is
%% given in the file 'manifest.txt'.
%%
 \message{File ignored}
\endinput
%%
%% End of file '.tex'.
```

4. Conclusiones

Luego de realizar el trabajo se pudieron rescatar algunos aspectos claves sobre hilos, procesos y señales. Respecto a los hilos es que dan la capacidad de tener más de una camino de ejecución de un mismo programa, luego los procesos permiten preservar el estado de un programa en un determinado momento y las señales poder enviar información a un programa en ejecución en tiempo real.

Referencias

[1] *Thread Arguments and Return Values*. URL: https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/ThreadArgs.html.