

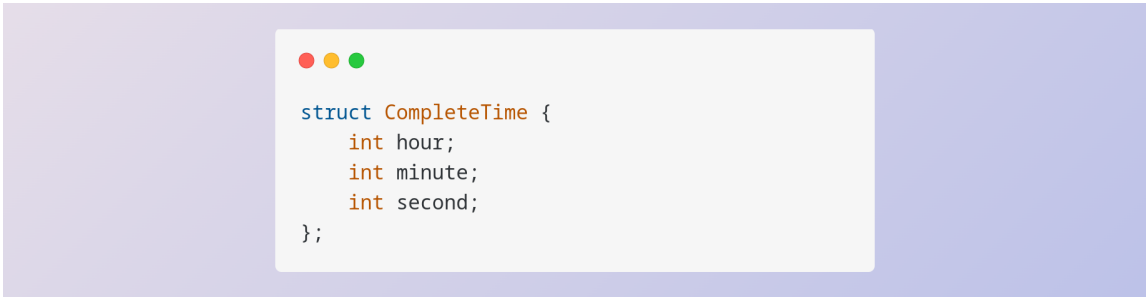
Reporte: Hilos Contra Procesos

Diego Ruiz Mora — 2202000335

29-Septiembre-2023

1 Funciones generalizadas para los procesos

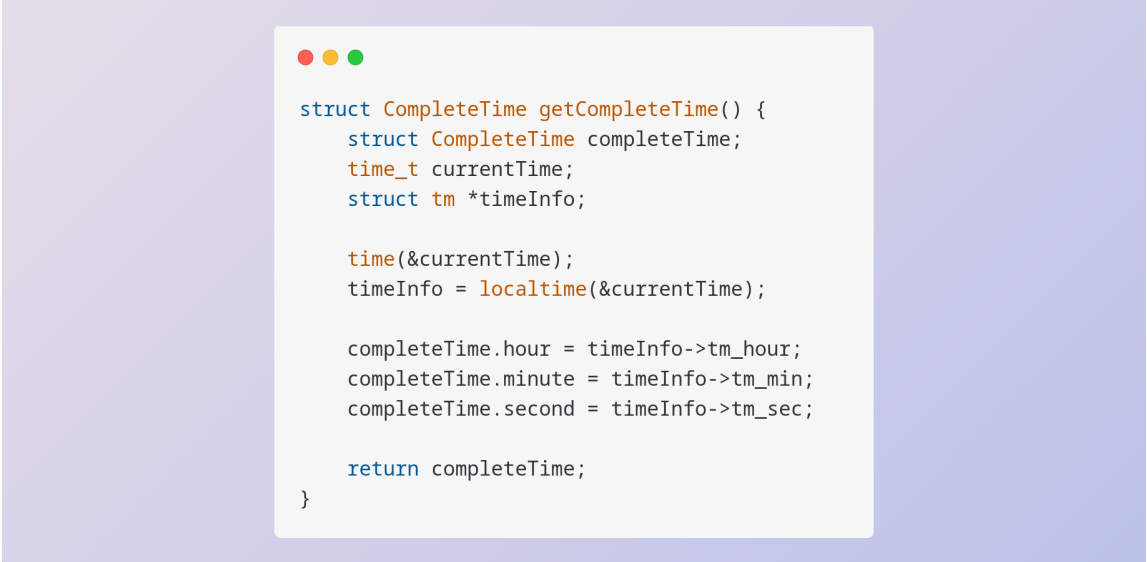
Comenzaremos dando las funciones que nos permitan hacer operaciones en común dentro de los tres programas principales. Solamente necesitamos una función que nos permita obtener la hora, por otro lado una función que nos permita escribir dentro de un archivo, y una sencilla estructura para recuperar la hora. Por comodidad observaremos la estructura, donde tenemos tres enteros que representaran las horas, minutos y segundos: Esto nos permitirá obtener la función que calcula la



```
struct CompleteTime {  
    int hour;  
    int minute;  
    int second;  
};
```

Imagen 1: Estructura Para Recuperar el Tiempo

hora completa, con minutos y segundos. Esta función no recibe nada y regresa como valor un *struct CompleteTime* como el que definimos momentos antes. En esta función veremos que tenemos que crear elementos del tipo *time_t* y otro apuntador a un dato *tm* ambos está contenidos dentro de la librería *time.h*. Seguido a ello, con el uso de la función **time** y la función **localtim()** ambas reciben un la referencia del elemento *currentTime*, de esta manera regresamos los valores asignándolos a un elemento del tipo de regreso de la función.



```

struct CompleteTime getCompleteTime() {
    struct CompleteTime completeTime;
    time_t currentTime;
    struct tm *timeInfo;

    time(&currentTime);
    timeInfo = localtime(&currentTime);


    completeTime.hour = timeInfo->tm_hour;
    completeTime.minute = timeInfo->tm_min;
    completeTime.second = timeInfo->tm_sec;

    return completeTime;
}

```

Imagen 2: Función Para Recuperar el Tiempo

Por último veremos la función que nos permite escribir en un archivo, donde tendremos como parámetros dos apuntadores a un elemento del tipo **char *** que justamente nos indica la posición de inicio de dos cadenas ; el nombre del archivo y la cadena que queremos escribir dentro del archivo. Creamos un apuntador del tipo *FILE* que nos permitirá recorrer las posiciones dentro del archivo, inicializándolo con la función **fopen()** determinar el nombre del archivo y la forma en la que abriremos este archivo. En caso de que este apuntador sea distinto de **NULL** nos permitirá escribir en el, con la función **fprintf()** y cerraremos la cadena de escritura con la función **fclose()**.



```

void escribir(char *nombreA, char *cadenaAEscribir) {
    FILE *filePointer = fopen(nombreA, "w");

    if (filePointer == NULL) {
        printf("\nEl archivo no pudo ser leído");
        return;
    }
    fprintf(filePointer, "%s\n", cadenaAEscribir);
    fclose(filePointer);
}

```

Imagen 3: Función Para Escribir En Un Archivo

2 Programa de creación de procesos

```
int main() {
    clock_t start_time, end_time;
    char hora[15];

    char nombreA[20];
    char cadenaAEscribir[60];

    struct CompleteTime timeNow;

    start_time = clock();

    for (int i = 0; i < 20; i++) {
        pid_t pid = fork();
        if (pid == 0) {
            timeNow=getCompleteTime();
            // Codigo del hijo
            sprintf(hora,"%02d:%02d:%02d", timeNow.hour, timeNow.minute, timeNow.second);
            sprintf(cadenaAEscribir,"%d. PID: %d; PPID: %d; Hora: %s",i, (int) getpid(), (int) getppid(), hora);

            sprintf(nombreA, "hijos1/hijo%d.txt",i+1);
            escribir(nombreA, cadenaAEscribir);

            exit(0);
        } else if (pid < 0) {
            perror("Fork failed");
            exit(1);
        }
    }

    end_time = clock();

    wait(NULL);

    double elapsed_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("\nTiempo total para crear 20 procesos: %.4f seg\n", elapsed_time);

    return 0;
}
```

Imagen 4: Programa Para Crear 20 Procesos

Notaremos que la función tiene bastante puntos a destacar. Primero que nada tendremos que crear unos datos del tipo *clock_t* que nos permitirá obtener el punto de inicio y de termino de programa para poder desplegar el tiempo empleado en la creación de los veinte procesos. Así mismo un arreglo de caracteres que nos ayudará a escribir la hora que obtenemos para cada archivo. Luego declararemos otros dos arreglos de carácter para la creación del archivo. Seguido de ello un *struct CompleteTime* que nos permite obtener la hora en un punto exacto. Seguido comenzamos un ciclo que se repita veinte veces, dentro de el declaramos una variable del tipo *pid_t* para poder hacer la llamada a sistema **fork()** para duplicar el proceso, lo cual ocurre justo después. Esto nos permite diferenciar entre el proceso hijo el proceso padre. En caso de ser uno de los hijos deberemos primero

que nada tomar el tiempo en ese momento y escribirla en la cadena de la hora.

Luego tendremos que escribir en la cadena que pasaremos al archivo. La cual contiene el identificador del proceso, el identificador del proceso padre y la hora a la que fue creado este hijo. Después de ello escribimos en la cadena que será el nombre del archivo, que es justo la siguiente instrucción mandando a llamar a la función que escribe. Terminamos esta parte con una salida exitosa.

Marcamos el tiempo de finalización de la creación y terminación de procesos para poco después hacer la diferencia, el cociente con un tipo de datos para obtener el dato en segundos y *castearla* a un *double* para visualizarlo mediante una impresión.

Pero antes de dar los resultados, podríamos hacer una pequeña pausa, solamente la utilizamos para lograr visualizar en las herramientas los 20 procesos que se crearon. Que es algo parecida a la idea de la llamada de sistema **wait(NULL)** para esperar a los procesos hijos a que concluyan.

3 Programa de creación de hilos

3.1 Función imagen para los hilos creados



Imagen 5: Función Imagen Para Los Hilos

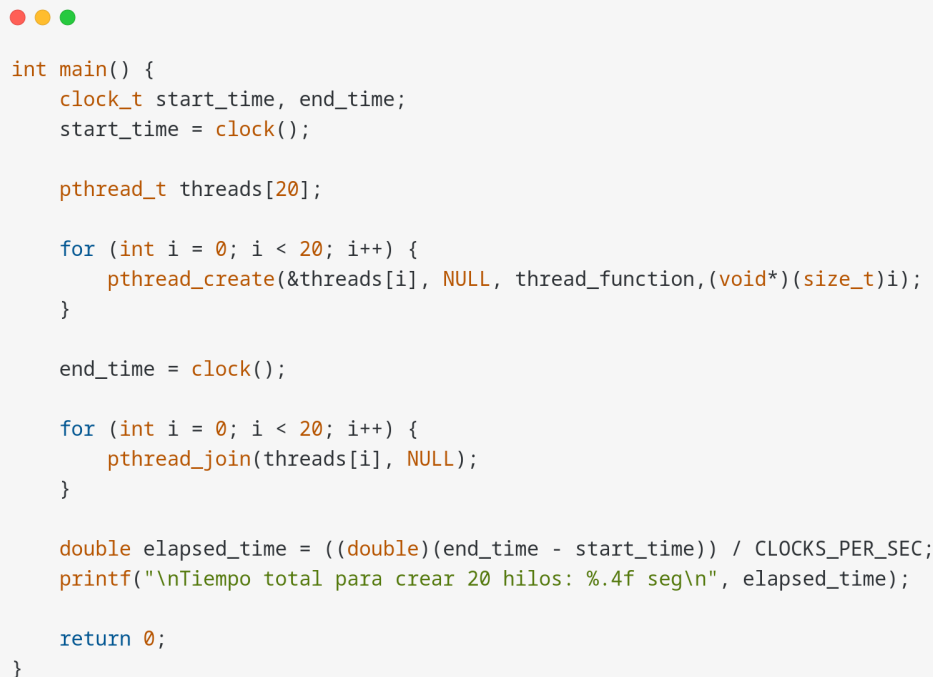
Como bien sabemos tenemos que crear una función que reciba un apuntador a nulo y que regrese un apuntador a nulo, que es justamente el caso que tenemos con esta función. Empezamos con una llamada a sistema **sleep()** que puede ir o no y que tiene la intención de visualizar los hilos con distintas herramientas.

Luego obtendremos los parámetros que en este caso será solamente el número de hilo que se creó, también tenemos que declarar un elemento del tipo *pthread_t* inicializandolo con el valor del identificador del hilo. Además dos cadenas, ambas para escribir en la cadena, tal como lo vimos en el

caso pasado y por ultimo un arreglo de caracteres para la hora.

De hecho podemos notar que el proceso para encontrar la hora en el momento exacto es lo mismo, por tal razón omitiremos la explicación del mismo. Después escribiremos en la cadenas para la generación del archivo y por ultimo **pthread_exit(0)** para informar que el hilo termina de manera correcta.

3.2 Función principal para la creación de hilos



```
int main() {
    clock_t start_time, end_time;
    start_time = clock();

    pthread_t threads[20];

    for (int i = 0; i < 20; i++) {
        pthread_create(&threads[i], NULL, thread_function, (void*)(size_t)i);
    }

    end_time = clock();

    for (int i = 0; i < 20; i++) {
        pthread_join(threads[i], NULL);
    }

    double elapsed_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("\nTiempo total para crear 20 hilos: %.4f seg\n", elapsed_time);

    return 0;
}
```

Imagen 6: Programa Para Crear 20 Hilos

Notaremos que al igual que en el caso pasado tenemos dos elementos que nos permitirán obtener el inicio y fin de la creación. De hecho, comenzamos con eso, inicializando la variable que nos marca el comienzo de la creación de hilos. Luego declaramos un arreglo de elementos del tipo *pthread_t* que nos permite diferenciar a cada uno de los hilos.

Para la creación de los 20 hilos usaremos un ciclo y dentro de el la instrucción **pthread_create()** con argumentos la referencia al arreglo de identificadores de hilos, *NULL* para el caso de la estructura que inicializa los hilos; osea la que es por defecto, la función que le da imagen al hilo y por ultimo los argumentos que recibirá el hilo, que como bien se menciono será el numero de hilo

creado. Justo en este punto es donde debemos de obtener el tiempo de finalización, justo después de terminar de crear los hilos.

Mas abajo haremos uso de otro ciclo para repetir la instrucción **pthread_join** que nos permite esperar a que los hilos se terminan de ejecutar. Esta función recibe el identificador de hilo y un *NULL* que son los valores que recuperaremos al termino del hilo. Terminamos justamente con lo mismo que en el caso pasado, el cálculo y la impresión del total de tiempo de ejecución.

3.3 Programa creación de hilos en procesos

En esta parte lo optimo sera crear los procesos con una estructura en abanico para luego darle código a cada uno de ellos, la idea principal es crear ciclo que cree 20 procesos y luego dentro de cada uno de estos procesos otro ciclo para crear cada uno de los hilos que necesitamos.

3.3.1 Función de la que toma la imagen los hilos

A screenshot of a code editor with a light blue background and a purple border. The code is written in C and defines a thread function. It includes comments in Spanish and uses standard C headers like pthread.h and time.h. The function obtains a thread ID, declares character arrays for a name and a string to write, gets the current time, and prints some information to the console. It also writes the thread ID, process ID, and time to a file named 'hijos3/P'.


```
void* thread_function(void* arg) {
    // Obtencion de id de hilo
    pthread_t tid = pthread_self();
    //Declaración de cadenas a utilizar
    char nombreA[20];
    char cadenaAEscribir[60];

    //Obtencion de hora
    char hora[15];
    struct CompleteTime timeNow;
    timeNow=getCompleteTime();
    sprintf(hora,"%02d:%02d:%02d", timeNow.hour, timeNow.minute, timeNow.second);

    //Escritura de variables para archivo
    sprintf(cadenaAEscribir, "\nID del hilo: %lu; PID: %d; Hora: %s", (unsigned long)tid, (int) getpid(), hora);
    sprintf(nombreA, "hijos3/P:%dT:%d", getpid(), (unsigned long) tid);
    escribir(nombreA, cadenaAEscribir);
    pthread_exit(0);
}
```

Imagen 7: Función Imagen Para Los Hilos

Al momento de implementar el programa haremos uso de una versión distinta de la función que nos permite la creación de los hilos, ya que en este caso puede ocurrir que se sobrescriban los datos si es que mandamos los contadores que utilizamos en los ciclos para las llamadas a la creación de procesos o hilos. Por lo que lo mejor opción es que al crear el archivo donde escribiremos incorporar el numero de hilo y el numero del proceso, lo que nos permitirá no tener elementos duplicados. Por otro lado en la función principal si que hay algunos cambios, pero realmente son pocos, ya que solo incorporamos el código para crear hilos dentro del código que ejecuta cada uno de los procesos creados. Como bien mencionamos para ello se utilizan dos ciclos, uno que nos permite crear cada uno de los procesos y dentro de este mismo otro ciclo que nos permite crear los 20 hilos por proceso.



```

int main() {
    clock_t start_time, end_time;

    start_time = clock();

    for (int i = 0; i < 20; i++) {
        pid_t pid = fork();
        if (pid == 0) {
            pthread_t threads[20];
            for (int j = 0; j < 20; j++) {
                pthread_create(&threads[j], NULL, thread_function, NULL);
            }

            for (int j = 0; j < 20; j++) {
                pthread_join(threads[j], NULL);
            }

            end_time = clock();
            exit(0);
        } else if (pid < 0) {
            perror("Fork failed");
            exit(1);
        }
    }

    end_time = clock();
    wait(NULL);

    double elapsed_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("Tiempo total para crear 20 procesos con 20 hilos cada uno: %.4f seg\n", elapsed_time);

    return 0;
}

```

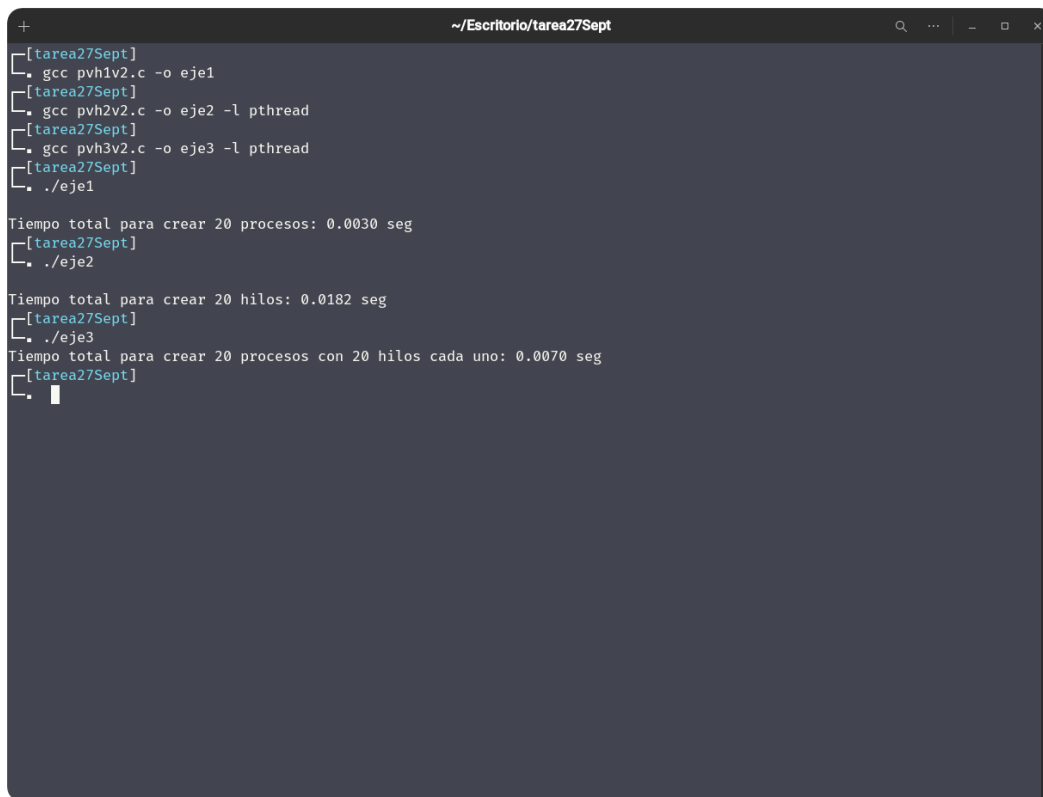
Imagen 8: Función que crea los procesos que a su vez crean los hilos

Fijemos la atención en la que a cada hilo no le pasamos absolutamente ningún argumento, ya que realmente no lo necesitamos como bien lo mencionamos, ya que para cada hilo lo mejor será crear los documentos con el identificador del proceso y el identificador del hilo. Obviamente que tenemos que crear las instrucciones que esperen cada uno de los hilos y a los procesos hijos, no colocaremos antes de ello la instrucción que nos marca la finalización de la creación porque se sobrescribe y en base a la experiencia arroja cantidades negativas de tiempo.

3.4 Resultados

Con base en los resultados notaremos que los tiempos quedan como a continuacion, empezando por el que menos tarda, al que mas tarda:

1. 20 Procesos
2. 20 Procesos con 20 Hilos
3. 20 Hilos.



```
+ ~/Escritorio/tarea27Sept
[tarea27Sept]
└─ gcc pvh1v2.c -o eje1
[tarea27Sept]
└─ gcc pvh2v2.c -o eje2 -l pthread
[tarea27Sept]
└─ gcc pvh3v2.c -o eje3 -l pthread
[tarea27Sept]
└─ ./eje1

Tiempo total para crear 20 procesos: 0.0030 seg
[tarea27Sept]
└─ ./eje2

Tiempo total para crear 20 hilos: 0.0182 seg
[tarea27Sept]
└─ ./eje3

Tiempo total para crear 20 procesos con 20 hilos cada uno: 0.0070 seg
[tarea27Sept]
└─
```

Imagen 9: Resultados

1. ¿Cuál de los dos programas es más rápido? Explique por qué sucede esto.
El de la creación de los procesos debido a que mi computadora tiene un procesador de 4 núcleos, lo que significa que hace 4 tareas de manera simultanea, cosa contraria a los hilos que no son al mismo tiempo, si no mas bien simulando esta situación, si bien con mas hilos, pero no se compara.

2. ¿Podría obligar a que los procesos en el programa hilosvsprocesos1.c escribieran sobre el mismo archivo? ¿Cómo haría eso? Indique las modificaciones necesarias al código
 Se podría considerando que no estamos diciendo que lo hagan de manera simultanea, por lo que cada proceso podría abrir el archivo, escribir en el y luego cerrarlo, para que el otro haga lo mismo, incluso creo que hay una función que bloquea o pausa la escritura en el proceso. Igualmente podríamos hacer un pipe para simular que los dos están escribiendo aunque en realidad solo uno lo hace., esto haciendo pasar las cadenas que quiera escribir al que si esta escribiendo. Del todo no se como realizar el código, pero se que se puede realizar de estas maneras.
3. ¿Podría obligar a que los hilos en el programa hilosvsprocesos2.c escribieran sobre el mismo archivo? ¿Cómo haría eso? Indique las modificaciones necesarias al código
 Si y a mi parecer es mas sencillo de hacer, recordemos que los hilos comparte los archivos abiertos, por lo que solamente tendríamos que generar un mecanismo que sincronice la escritura o para que lo hagan en lineas distintas., incluso podríamos guardar la escritura en un buffer temporal que permita simular que se hace de manera simultanea.
4. En ambos casos ¿Existiría algún problema? Justifique se respuesta.
 Si, claro que existen problemas, sobre todo el generar los mecanismos de sincronizacion que quiza no sean de lo mas sencillo, otro caso claro, es que podamos sobrecribir los datos que ahi se encuentra, otro problema que tendremos es el desorden en el que pueden llegar las cadenas al archivo, ya que como hemos visto antes, ocurre que unos procesos se ejecutan antes que otros.
5. ¿Por qué es importante que tanto los hilos como los procesos compartan información?
 La respuesta obvia es la respuesta anterior, porque resulta que si queremos dividir el trabajo en distintas tareas, algunas de ellas se necesitaran comunicar para generar una sincronizacion entre todos los procesos, lo que nos hará evitar colisiones entre los mismo. Y no solo eso, si no también para brindar información importante acerca del estado de finalización de un estado, esto puede desencadenar distintas rutas de ejecución o distintas creaciones de procesos. Las razones son variadas y múltiples.