

Reporte: Hilos Contra Procesos

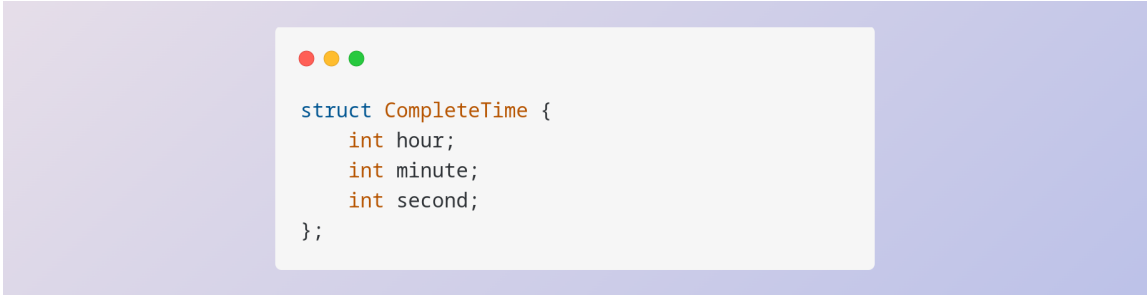
Diego Ruiz Mora — 2202000335

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.

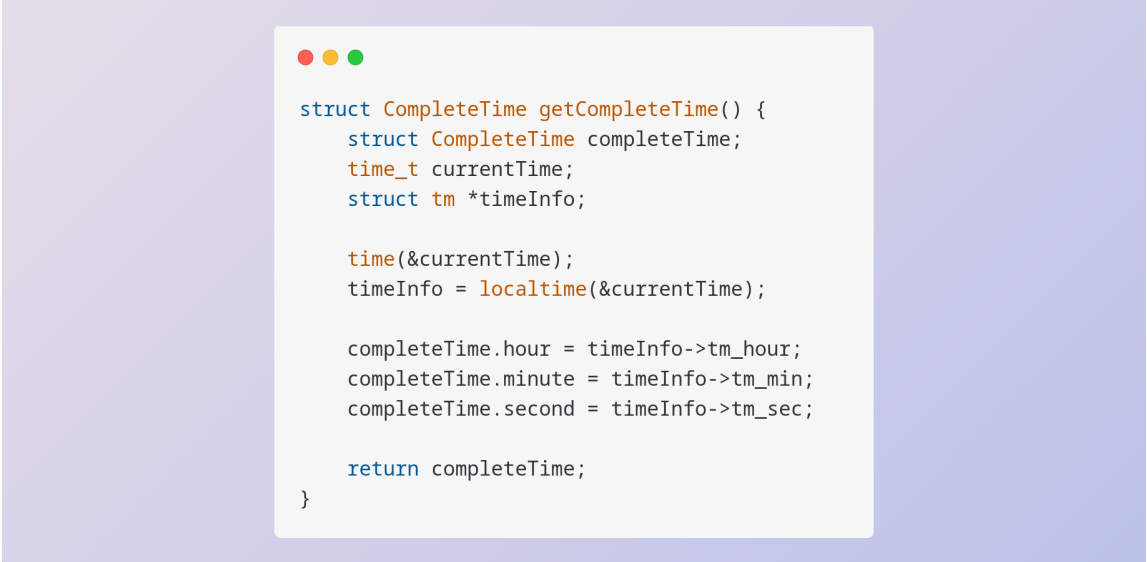
Por comodidad observaremos la estructura, donde tenemos tres enteros que representaran las horas, minutos y segundos: Esto nos permitirá obtener la función que calcula la hora completa, con

A screenshot of a code editor window with a light gray background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in a dark blue font and defines a C struct named CompleteTime with three integer members: hour, minute, and second.

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

Imagen 1: Estructura Para Recuperar el Tiempo

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;

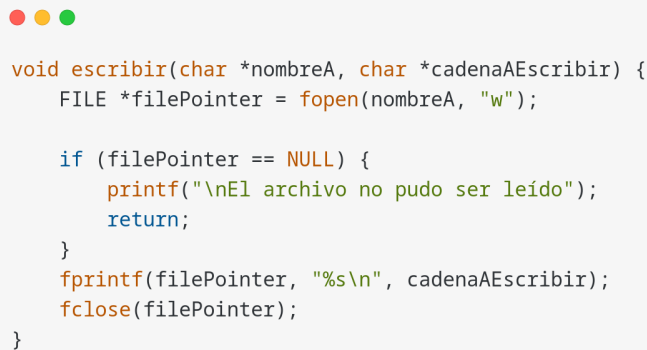
    return completeTime;
}

```

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

A screenshot of a code editor with a light blue background and a purple border. The editor shows a C program that creates 20 child processes using `fork()`. The program includes headers for `clock_t`, `pid_t`, `time`, and `unistd.h`. It declares variables for start and end times, a character array for the hour, and arrays for the child's name and the string to be written. A `struct CompleteTime` is used to store the current time. The main function starts by recording the start time, then enters a loop that forks 20 children. Each child writes its PID, PPID, and the current time to a file named `hijos1/hijo%d.txt`. The parent process waits for all children to finish using `wait(NULL)` and then calculates the total time taken. The code is as follows:

```
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);
        }
    }

    sleep(15);
    wait(NULL);

    end_time = clock();

    double elapsed_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("Total time taken to create and wait for 20 processes: %.4f seconds\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.

Luego haremos una pequeña pausa, que se puede omitir, 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.

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.

3 Programa de creación de hilos

3.1 Función imagen para los hilos creados



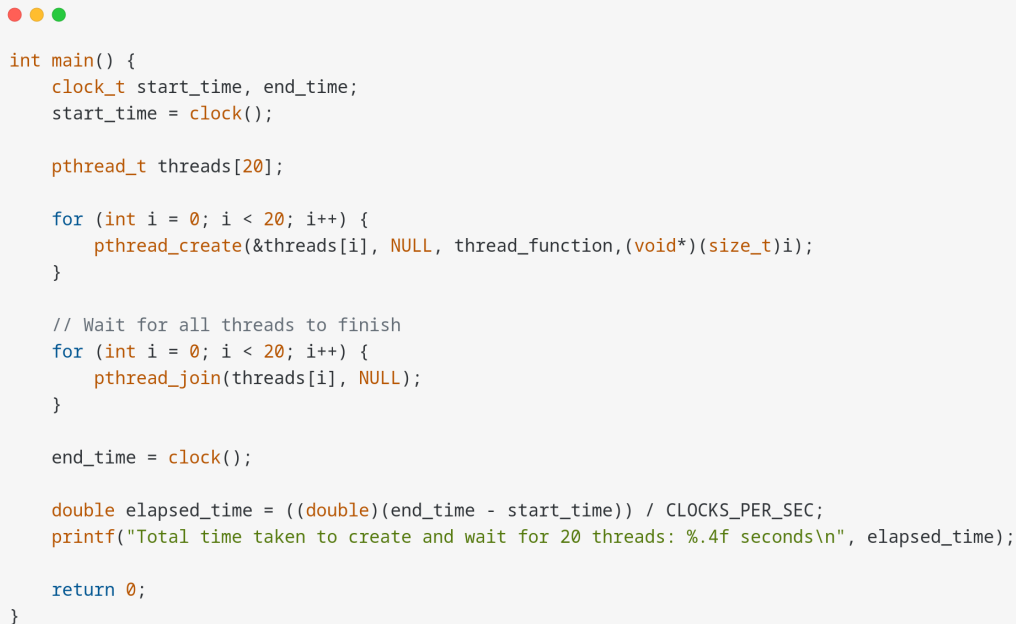
Imagen 5: Función Imagen Para Los Hilos

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);
    }

    // Wait for all threads to finish
    for (int i = 0; i < 20; i++) {
        pthread_join(threads[i], NULL);
    }

    end_time = clock();

    double elapsed_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("Total time taken to create and wait for 20 threads: %.4f seconds\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 estruc-

tura 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.

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.