Universidad De Magallanes

Facultad De Ingeniería

Departamento De Ingeniería En Computación

Laboratorio De Introducción a Sistemas Operativos

Práctica No° 001

Creación y manejo de procesos

Estudiante: Milton Hernández - milherna@umag.cl

Fecha y hora: 30 de Septiembre del 2024- 10:00 Pm

Profesor: Eduardo Peña J.

Asignatura: Introducción a Sistemas Operativos

Status del documento

Práctica N°001 **Creación y manejo de procesos**			
Número de referencia del documento			001
Versión	**Revisión**	**Fecha**	**Razones del cambio**
1.0	1	30 de Septiembre del 2024	Primera revisión del documento

Registro de cambios del documento		**RCD N°:**	001
		Fecha:	30 de Septiembre del 2024
		Otiginado por:	Milton Hernández M.
		Aprobado por:	Milton Hernández M.
Titulo del documento:		Práctica N°001 Creación y manejo de procesos	
Número de referencia del documento:			001
Versión del documento	1.0	**Número de revisión:**	1
Página	**Párrafo**	**Razones del cambio**	

Índice

Resumen

Conocer el manejo de procesos dentro de un sistema operativo es fundamental para comenzar a trabajar con aplicaciones que saquen provecho de las capacidades que ofrecen los sistemas computacionales modernos. En el presente informe se busca explorar algunos de los conceptos relacionados con la ejecución de procesos así como repasar conceptos básicos de programación en C que pueden ser útiles para el desarrollo de aplicaciones que faciliten la comprensión del uso de procesos.

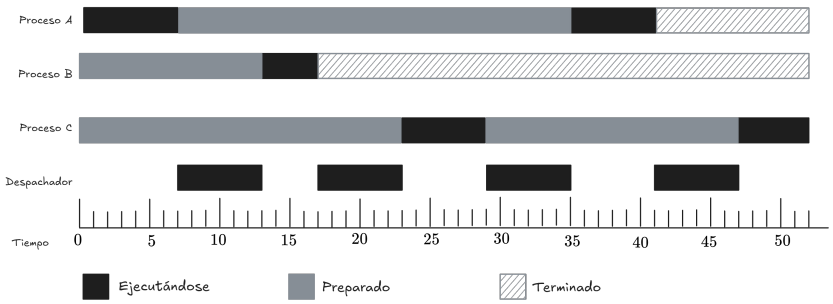
1. Introducción

Un proceso se define como un programa en ejecución dentro de un sistema operativo. Los procesos conforman la mayoría de las acciones que se realizan dentro de un sistema operativo, desde la las entradas y salidas del mismo, la creación o eliminación de archivos y carpetas, hasta la ejecución de programas más complejos.

Inicialmente dentro del mundo de la computación los sistemas operativos trabajaban un proceso después de otro de manera secuencial, es decir, un proceso debía terminar por completo para que el siguiente empezara a ejecutarse. Sin embargo, posteriormente se avanzó en este sentido consiguiendo que los sistemas operativos trabajen los procesos en paralelo, esto quiere decir que la CPU (Unidad central de procesamiento) atiende a varios procesos al mismo tiempo.

«Todas las computadoras modernas pueden hacer varias cosas al mismo tiempo. Mientras ejecuta un programa de usuario, una computadora también puede estar leyendo de un disco y enviando texto a una pantalla o impresora. En un sistema de multiprogramación, la CPU también conmuta de un programa a otro, ejecutando cada uno durante decenas o centenas de milisegundos. Si bien, estrictamente hablando, en un instante dado la CPU está ejecutando sólo un programa, en el curso de un segundo puede trabajar con varios programas, dando a los usuarios la ilusión de paralelismo.» (Tanenbaum & Woodhull, 1997). En este sentido, los sistemas operativos trabajan procesos en paralelo, pero en una CPU única tal como se ilustra en la figura 1.

Figura 1: Procesos en paralelo



Esto ocasiona que al querer trabajar con procesos de manera paralela (Por ejemplo, dejar un proceso en segundo plano)

sea necesario tener en cuenta este comportamiento.

Adicionalmente, para la compresión del funcionamiento de procesos dentro de un sistema operativo, es necesario poder hacer experimentos con los mismos, lo que se hará en el presente laboratorio mediante el uso de procesos productores (generadores de información) y temporizadores que nos permitirán comprender su funcionamiento y tiempos de acción.

2. Objetivo Principal

Experimentar, mediante la creación de programas productores de elementos aleatorios el funcionamiento básico de los procesos concurrentes dentro de un Sistema Operativo.

3. Objetivos secundarios

1. Reforzar conocimientos del lenguaje C como medio para la creación de programas relacionados con la gestión de procesos dentro de Un Sistema Operativo.
2. Indagar la manera en que se ejecutan los procesos dentro de Un Sistema Operativo, y las maneras que existen para eliminar un proceso en medio de su ejecución.

4. Marco teórico

El marco teórico se divide en tres secciones: la primera describe el funcionamiento de los procesos dentro de un sistema operativo, y la segunda aborda la creación de procesos concurrentes y la tercera ahondará un poco en la creación de objetos aleatorios dentro de un sistema computarizado.

4.1. Funcionamiento de los procesos dentro de un sistema operativo

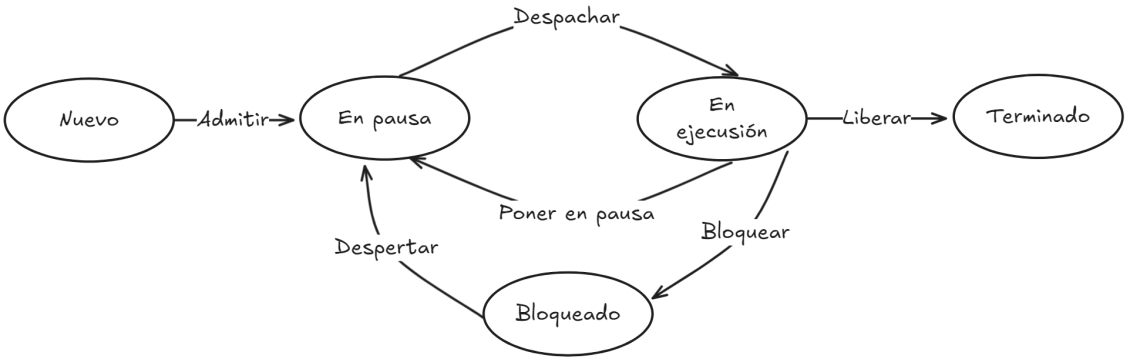
Como hemos mencionado anteriormente los procesos se pueden definir como programas que se ejecutan dentro de un sistema operativo.

Cabe mencionar que un proceso puede pasar por distintos estados durante su ejecución, algunos de estos estados son:

- **Proceso nuevo:** Cuando un proceso es creado por algún medio este se encuentra en estado nuevo.
- **En ejecución:** Un proceso está en ejecución cuando la CPU se encuentra procesando información a su respecto en un instante determinado de tiempo.
- **En pausa:** cuando el proceso es suspendido temporalmente por el sistema operativo este se pone en pausa, a la espera de poder continuar su ejecución. (Estas pausas pueden llegar a durar muy pocos milisegundos).
- **Bloqueado:** se refiere a cuando un proceso está esperando que ocurra un evento externo, como la disponibilidad de un recurso. En este estado el proceso puede durar más tiempo que en una pausa e incluso puede no volver a ejecutarse.
- **Terminado:** Cuando todas las tareas relacionadas con un proceso se han completado este proceso se encuentra en estado terminado.

Podemos ver un gráfico del comportamiento de estos estados en la figura 2.

Figura 2: Estados de los procesos



Comprender los diferentes estados de un proceso así como el hecho de que al ejecutar en un mismo periodo varios procesos lo que ocurre realmente es que la CPU se encarga de ejecutar cada uno de ellos en un corto periodo en un orden determinado. (Como explicado en la introducción) es el fundamento del presente laboratorio.

4.2. Creación de procesos concurrentes dentro de un sistema operativo

Trabajar con procesos concurrentes dentro de un sistema operativo puede ser complejo si no se cuenta con los conocimientos necesarios para su creación y manejo.

Supongamos que, como se realizará posteriormente, tenemos tres programas encargados de la generación de productos de manera continua (números, letras, etc) y que deseamos ejecutarlos de manera paralela o, como se mencionará en adelante **de manera distendida**.

En caso de trabajar en un sistema Linux (como es el caso del presente laboratorio) para ejecutar los procesos en segundo plano, se puede usar el comando &, lo que permite que el proceso continúe su ejecución mientras la consola queda disponible para otras tareas.

```
$ ./Generador_uno &
$ ./Generador_dos &
$ ./Generador_tres &
```

Esto indica que el proceso generador_uno se ejecutará en segundo plano, lo mismo para el generador_2 y para el generador_tres. Al final de esta ejecución la consola quedará libre para poder continuar con otras tareas.

Para poder ver los procesos en ejecución y eliminarlos es necesario utilizar el comando *ps* que nos devuelve una lista de todos los procesos en ejecución del sistema operativo.

```
$ ps
  PID TTY          TIME CMD
  101 pts/0    00:00:00 bash
  102 pts/0    00:00:00 ps
```

«*Si introduces el comando ps sin opciones, únicamente te mostrará los procesos iniciados por el shell actual. Por lo tanto, otros procesos quedan excluidos inicialmente.*» (IONOS, 2023).

Durante el presente informe se realizarán varios programas para su posterior ejecución mediante el uso del lenguaje C como veremos en el Procedimiento Experimental.

4.3. Creación de números aleatorios

«*Por naturaleza, las computadoras no son muy buenas con el azar. Su fortaleza radica en generar salidas predecibles ejecutando secuencias de operaciones programadas. Eso es casi lo opuesto a la aleatoriedad.*» (Academy, s.f.).

De acuerdo con de Cabezón (2022) el mejor método de generación de números aleatorios se encuentra en los sistemas cuánticos como el usado por la web random.org. Sin embargo no es muy cómodo tener que recurrir a información externa a un equipo para generar números aleatorios, por lo cuál se crearon con el tiempo algoritmos que generan números llamados **Pseudo Aleatorios**, es decir números que detrás llevan un desarrollo no aleatorio pero que a los ojos de los comsumidores lo son.

Una de las maneras de generar números de este tipo es mediante la técnica **LFSR** que se traduce al español como: *registro de desplazamiento con retroalimentación lineal* que utiliza funciones lógicas sencillas y por tanto es muy sencilla de programar y de implementar a nivel del Hardware. El proceso consiste en lo siguiente:

1. Se inicia con una semilla en binario.
2. En cada repetición del proceso de cálculo se saca el último bit del número semilla.
3. Posteriormente el resto de bits se corren a la derecha.
4. En el espacio sobrante se agrega un bit, resultado de una operación lógica realizada sobre los bits restantes.
5. Este proceso se repite, generando con los bits que se van retirando un número binario que se puede entregar como äleatorio.

Como se ve el procedimiento depende de la semilla usada; en caso de usar la misma semilla los resultados serán iguales, por lo que se precisa de una manera certera de crear semillas con las que alimentar este generador de pseudo-aleatoriedad.

Los diferentes lenguajes de programación tienen diferentes maneras de conseguir la semilla que será usada, como por ejemplo:

- Usar el tiempo actual del sistema en el momento.
- Usar la cantidad de procesos activos en el sistema en el momento.
- Usar la cantidad de bytes libres en memoria en el momento.
- Usar el número del proceso en ejecución.

5. Procedimiento experimental

El Procedimiento experimental propuesto para el presente laboratorio es el siguiente:

1. Crear tres procesos generadores de palabras.
   - Primer proceso Genera números enteros aleatorios entre 0 y 100.
   - Segundo proceso Genera letras aleatorias de la A a la Z.
   - Tercer proceso Genera una letra acompañada de un numero entero.
2. Para cada generador se debe considerar que tras cada creación debe haber un tiempo de retardo delay() entre 0 y 3 seg
3. Determinar el tiempo de ejecución de cada generador de palabras.
4. Correr los tres generadores al mismo tiempo en forma desatendida, liberando así la consola de comandos.
5. Guardar los datos generados en tiempo de ejecución de cada uno de los tres generadores.
6. Si el generador es infinito, ver; Número de proceso en ejecución y eliminación del proceso.

Mediante la realización del presente informe buscaremos responder las siguientes preguntas relacionadas con los procesos:

1. De que manera se puede realizar verdadera aleatoriedad de un numero?
2. Si los tres generadores crean 10.000 palabras cada uno, demoran el mismo tiempo de ejecución?
3. Si se crean 50 generadores idénticos, el tiempo de ejecución es el mismo para todos?
4. De que manera se pueden eliminar todos los procesos en ejecución de una sola vez
5. Cuál es el comando que elimina procesos?
6. Si quitemos los delay de tiempos, la ejecución de los generadores será la misma? si no es así, a que se debe?
7. Qué otros procesos ejecuta el Sistema Operativo mientras se ejecutan los que que generamos?

6. Datos obtenidos

La primera tarea que se debe realzar es la creación de funciones que, generen números y letras aleatorias. Para esto se hizo uso de la biblioteca 'stdlib.h' que nos permite acceder a la función 'rand()' que nos devuelve un número entero entre 0 y 32767 dada una semilla que se le pasa como parámetro.

Teniendo esto en cuenta generamos las funciones 'randomInt' y 'randomLetter' que se pueden ver en el anexo 1.

Adicionalmente se creó la función 'delay()' (Que se ve en el anexo 2) que recibe un tiempo mínimo y uno máximo y realiza un bucle que dura un tiempo aleatorio entre los dos entregados (generando un tiempo en que el proceso está "Pausado").

Luego de crear las funciones que generan los resultados aleatorios el siguiente paso es crear los tres procesos solicitados. Para esto se pasó por las siguientes fases.

1. Inicialmente se intentó crear procesos que generaran items y los entregaran por la salida estándar. Al hacer pruebas con esta idea se notó que no era posible guardar la información de loa archivos como se deseaba, puesto que se quería guardar la información en un archivo aparte y se intentó hacer con una redirection de la salida estándar, pero se tuvo dificultades para guardar la información en caso de detener el proceso en ún momento inesperado desde la consola.

2. Dado esto se decidió optar por abrir los archivos destino dentro del propio proceso, encontrando un error similar. El terminar la ejecución del proceso en ocasiones no se terminaba de guardar la información en el archivo.

3. Se optó entonces por vaciar el buffer del archivo luego de escribir cada uno de los datos generados dentro del mismo, para esto se usó la función 'fflush()' vacía el buffer de un archivo pasado como parámetro. El inconveniente que se encontró con este planteamiento fue que no era posible cerrar el archivo en caso de que el procesos fuera terminado forzosamente, pero se decidió usar el sistema logrado hasta este momento.

Con este proceso se consiguieron los procesos generadores que se encuentran en los anexos 3, 4 y 5 respectivamente. En los tres casos se crean distintos archivos que una vez ejecutados los programas pueden ser accedidos para conocer los resultados de los programas.

Ahora hace falta poder ejecutar los tres procesos de manera distendida para lo cuál se hizo uso de la terminal con el comando '&' para ejecutar procesos en segundo plano. La idea consiste en crear un cuarto programa llamado 'main.c' que ejecuta los tres procesos en segundo plano mediante llamadas al sistema. Pero no sólo basta con eso puesto que al llamar al sistema se crean los procesos de manera independiente, esto quiere decir que no dependen de la ejecusión de 'main.c', por lo tanto lo que se propuso fué que al ejecutar main.c se especifique un tiempo de ejecución tras el cuál los tres programas serán destruidos.

Para la destrucción de los procesos se hizo uso de la función del terminal 'killall' que, de acuerdo con Linux (2023)

envía una señal (siendo por defecto una para terminarlos) a los procesos que se mencionen luego del comando.

De esta manera se obtiene el resultado del archivo ‘main.c’ presente en el anexo 6 que ejecuta los tres procesos en segundo plano y que se puede ejecutar con el comando.

La salvedad importante de mencionar en este punto es que no se consiguió generar una salvaguarda para que cuando el main sea eliminado los otros procesos también lo sean.

A continuación se muestra un ejemplo con los resultados obtenidos luego de ejecutar todos los procesos durante 25 segundos (Las tablas son leidas de los archivos ‘.csv’ generados con los procesos y son interpretadas con el código python que se muestra en el anexo 7).

Tabla 1: Datos correspondientes al archivo programa1.csv luego de 25 segundos

Intem	Valor	Tiempo Partial	Tiempo acumulado
0	36	2.052	2.052
1	88	1.371	3.423
2	77	2.941	6.364
3	6	2.284	8.648
4	7	0.968	9.616
5	97	1.712	11.328
6	96	0.154	11.482
7	35	1.011	12.492
8	31	0.038	12.531
9	26	1.9	14.43
10	54	0.891	15.321
11	66	2.302	17.623
12	61	0.74	18.363
13	18	2.526	20.889
14	45	0.787	21.676
Datos obtenidos de programa			

Tabla 2: Datos correspondientes al archivo programa2.csv luego de 25 segundos

Intem	Valor	Tiempo Partial	Tiempo acumulado
0	a	1.21	1.21
1	R	0.482	1.692
2	F	1.429	3.121
3	p	2.785	5.907
4	b	1.109	7.015
5	d	2.604	9.62
6	X	2.921	12.541
7	C	0.752	13.293
8	I	0.26	13.553
9	D	1.373	14.926
10	T	1.526	16.452
11	o	1.062	17.514
12	Z	0.484	17.998
13	D	1.591	19.589
14	S	2.914	22.504
15	m	0.355	22.859
Datos obtenidos de programa			

Tabla 3: Datos correspondientes al archivo programa3.csv luego de 25 segundos

Intem	Valor	Tiempo Partial	Tiempo acumulado
0	a36	1.21	1.21
1	e59	2.941	4.151
2	p6	2.785	6.936
3	d35	2.604	9.54
4	P57	1.011	10.551
5	I31	0.26	10.811
6	i4	0.891	11.702
7	o66	1.062	12.764
8	C80	2.526	15.289
9	S45	2.914	18.204
10	l81	1.124	19.328
11	N6	1.065	20.393
12	u79	1.367	21.759
13	P13	0.078	21.837
Datos obtenidos de programa			

Posteriormente será necesario analizar estos datos, comprendiendo el significado de los tiempos obtenidos.

7. Análisis y discusión de los resultados

Una vez ejecutados los tres procesos podemos obtener resultados realmente relevantes de su comportamiento.

Luego de 25 segundos de ejecisoón de los tres procesos **de manera distendida** uno de ellos tardó 21.676 segundos, otro 22.859 segundos y el último 21.837 segundos, todos tiempos diferentes pero cercanos. Esto ocurre debido a que todos los delays están en el mismo rango de tiempos (0 a 3 segundos).

La pregunta sería qué ocurriría si suspendemos los delays. ¿Obtendríamos los mismos tiempos?. Lo más probable es que los tiempos sean similares, puesto que no hay delay alguno, pero no serían iguales puesto que el tiempo de ejecución de cada proceso es diferente (El tiempo que dedica la CPU a la ejecución de cada uno de los procesos).

Además es importante recalcar un hecho importante respecto a la generación de números aleatorios y esta tiene que ver con la semilla usada. En todos los procesos mostrados en el informe se usa como semilla el tiempo actual del sistema, pero antes del uso 'rand()' lo que se hace es generar un delay aleatorio y posterior a este volver a iniciar el generador de números aleatorios con el nuevo tiempo como semilla.

Ahora bien, si se suprimiera esta propuesta de semilla podría ocurrir que al ejecutar múltiples instancias del mismo proceso los resultados fueran los mismos puesto que la semilla es la misma al ser ejecutados (casi) al mismo tiempo.

Duranrte la realización de este experimento se comprendió el uso de comandos como 'kilall' para terminar procesos, o 'ps' para conocer aquellos en ejecución dentro del terminal actual; pero vale mencionar que existen otros procesos ejecutándose simultáneamoente a los ejecutados por el usuario, como se puede apreciar en el siguiente estracto de la terminal (Omitiendo parte de la salida entregada):

```
$ ./build/main.out &
[1] 58006
$ ps -A
  PID   TTY          IME CMD
    1     ?    00:01:25 systemd
49579 pts/4    00:00:05 node
49587 pts/2    00:00:01 node
49618 pts/2    00:01:45 node
49642 pts/2    00:00:00 node
49717 pts/2    00:00:00 node
49739 pts/2    00:00:00 cpptools
49787 pts/2    00:00:06 sm-agent
49810 pts/2    00:01:23 node
58006 pts/0    00:00:02 main.out
58008 pts/0    00:00:02 proceso1.out
58010 pts/0    00:00:02 proceso2.out
58012 pts/0    00:00:02 proceso3.out
58026 pts/0    00:00:00 ps
```

Además podemos destacar algunas dificultades que se presentaron durante la realización del experimento:

1. No se consiguió, con las herramientas disponibles, generar un sistema que permitiera que al eliminar el proceso principal se eliminaran los otros procesos generadores.
2. No se consiguió generar un salvaguarda para cerrar de manera adecuada los archivos abiertos por los procesos generadores en caso de que estos sean terminados de manera abrupta.
3. Al intentar abrir un mismo archivo con varios procesos al mismo tiempo, a priori no se presentaron errores, sin embargo, el archivo no queda correctamente escrito. Se puede atribuir este comportamiento a una [Situación de competencia] donde los procesos intentan acceder al mismo archivo al mismo tiempo y queda escrito en este solamente la información del último proceso que accede al archivo, perdiendo información en el camino.

8. Conclusiones y recomendaciones

Finalmente a raíz de lo realizado en este experimento se puede concluir lo siguiente:

1. Se pudo comprender de manera teórica el funcionamiento de los procesos concurrentes dentro de un Sistema Operativo.
2. Además se logró llevar a cabo la creación de tres procesos concurrentes que generan números aleatorios, letras aleatorias y letras con números, registrando sus resultados en archivos de manera ‘Simultánea’.
3. Se consiguió comprender el funcionamiento de la generación de números aleatorios dentro de sistemas computacionales, comprendiendo la naturaleza no aleatoria de los mismos y los métodos de generación de números pseudoaleatorios usados en los sistemas actuales, en particular el lenguaje C.

Los objetivos de este laboratorio se cumplieron a cabalad, sin embargo quedan pendientes diversas mejoras que se pueden realizar, como por ejemplo:

1. Comprender el funcionamiento de procesos padres e hijos permitiendo el manejo de procesos concurrentes de mejor manera.
2. Comprender el funcionamiento de la comunicación entre procesos, permitiendo enviar mensajes (más allá de lo aprendido con ‘killall’) y manejar los mensajes recibidos de manera adecuada.

9. Anexos

Listing 1: Archivo random.h

```c
#ifndef RANDOM
#define RANDOM
#include <stdlib.h>
#include <time.h>

int randomInt(int min, int max){
    return (rand()%(max+1-min))+min;
}

char randomLetter(){
    int min = 65; // A en ASCII
    int max = 122; // z en ASCII
    char letter;

    do
    {
        letter = (rand()%(max+1-min))+min;
    }while(letter>90 && letter<97); // Entre 91 y 96 no son letras

    return letter;
}

#endif
```

Listing 2: Archivo delay.h

```
#ifndef DELAY
#define DELAY
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

double delay(int min, int max)
{
  clock_t tiempo_inicio, tiempo_final, tiempo_espera;
  if(max)
  {
    clock_t minclock = min*CLOCKS_PER_SEC;
    clock_t maxclock = max*CLOCKS_PER_SEC;

    tiempo_espera =(rand()%maxclock+1-minclock)+minclock;
  }
  else
    tiempo_espera = min * CLOCKS_PER_SEC;

  tiempo_inicio = clock();
  while (clock() < tiempo_inicio + tiempo_espera);

  tiempo_final = clock();
  tiempo_final -= tiempo_inicio;

  return (double) tiempo_final/CLOCKS_PER_SEC; // Retornamos el tiempo de delay
}

#endif
```

Listing 3: Archivo proceso1.c

```c
#include <stdio.h>
#include "random.h"
#include "delay.h"

int main(){
    int element;
    int i=0;
    double execTime=0;
    double itemTime;

    FILE *data, *info;
    if((data= fopen("./build/dataproceso1.txt","w")) == NULL)
    {
        printf("No se pudo abrir el archivo ./build/dataproceso1.txt\n");
        exit(-1);
    }
    if((info= fopen("./build/infoproceso1.csv","w")) == NULL)
    {
        printf("No se pudo abrir el archivo ./build/infoproceso1.csv\n");
        exit(-1);
    }

    srand(time(0));
    delay(1,2);
    srand(rand());
    fprintf(info,"Item,Value,Partial,Total\n");
    while(1)
    {
        element = randomInt(0,100);
        //printf("%d\n", element);
        fprintf(data,"%d\n", element);
        fflush(data);
        itemTime = delay(0,3);
        execTime += itemTime;

        //printf("Proceso 1 - Item %d demoro: %.3lf segundos\n",i,  execTime);
        fprintf(info,"%d,%d,%.3lf,%.3lf\n",i,element,itemTime,execTime);
        fflush(info);
        i++;
    }

    fclose(data);
    fclose(info);
}
```

Listing 4: Archivo proceso2.c

```c
#include <stdio.h>
#include "random.h"
#include "delay.h"

int main(){
    int i=0;
    char element;
    double execTime=0;
    double itemTime;

    FILE *data, *info;
    if((data= fopen("./build/dataproceso2.txt","w")) == NULL)
    {
        printf("No se pudo abrir el archivo ./build/dataproceso2.txt\n");
        exit(-1);
    }
    if((info= fopen("./build/infoproceso2.csv","w")) == NULL)
    {
        printf("No se pudo abrir el archivo ./build/infoproceso2.csv\n");
        exit(-1);
    }

    srand(time(0));
    delay(1,2);
    srand(rand());
    fprintf(info,"Item,Value,Partial,Total\n");
    while(1)
    {
        element = randomLetter();
        //printf("%c\n", element);
        fprintf(data,"%c\n", element);
        fflush(data);
        itemTime = delay(0,3);
        execTime += itemTime;

        //printf("Proceso 2 - Item %d demoro: %.3lf segundos\n",i,  execTime);
        fprintf(info,"%d,%c,%.3lf,%.3lf\n",i,element,itemTime,execTime);
        fflush(info);
        i++;
    }

    fclose(data);
    fclose(info);
}
```

Listing 5: Archivo proceso3.c

```c
#include <stdio.h>
#include "random.h"
#include "delay.h"

int main(){
    int number;
    int i = 0;
    char letter;
    double execTime=0;
    double itemTime;

    FILE *data, *info;
    if((data= fopen("./build/dataproceso3.txt","w")) == NULL)
    {
        printf("No se pudo abrir el archivo ./build/dataproceso3.txt\n");
        exit(-1);
    }
    if((info= fopen("./build/infoproceso3.csv","w")) == NULL)
    {
        printf("No se pudo abrir el archivo ./build/infoproceso3.csv\n");
        exit(-1);
    }

    srand(time(0));
    delay(1,2);
    srand(rand());
    fprintf(info,"Item,Value,Partial,Total\n");
    while (1)
    {
        number = randomInt(0,100);
        letter = randomLetter();
        //printf("%c%d\n", letter,number);
        fprintf(data,"%c%d\n",letter,number);
        fflush(data);
        itemTime = delay(0,3);
        execTime += itemTime;

        //printf("Proceso 3 - Item %d demoro: %.3lf segundos\n",i,  execTime);
        fprintf(info,"%d,%c%d,%.3lf,%.3lf\n",i,letter,number,itemTime,execTime);
        fflush(info);
        i+=1;
    }

    fclose(data);
```

```
45      fclose(info);
46  }
```

Listing 6: Archivo main.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <time.h>
4
5   #include "delay.h"
6
7   int main(int argc, char *argv[]){
8       int tiempo_espera=0;
9       if(argv[1])
10          tiempo_espera = atoi(argv[1]);
11      if(!tiempo_espera)
12          tiempo_espera = 10;
13
14
15      if(system("./build/proceso1.out &"))
16          printf("Error al ejecutar proceso 1\n");
17      if(system("./build/proceso2.out &"))
18          printf("Error al ejecutar proceso 2\n");
19      if(system("./build/proceso3.out &"))
20          printf("Error al ejecutar proceso 3\n");
21
22      delay(tiempo_espera,0);
23
24      if(system("killall proceso1.out proceso2.out proceso3.out"))
25          printf("NO SE PUDO MATAR A LOS PROCECSOS!\n");
26      printf("Los procesos han terminado luego de %d segundos\n", tiempo_espera);
27
28      return 0;
29  }
```

Listing 7: Archivo tables.py

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os

# Formato para eliminar ceros finales en LaTex
def custom_format(x):
    # Not Cientifica
    formatted_str = ('%.6g' % x)

    if '.' in formatted_str:
        # Erase Final Ceros after the decimal point
        formatted_str = formatted_str.rstrip('0').rstrip('.')

    return formatted_str

def CreateLatexTable(csvFilePath, latexFilePath, programNum):
    data = pd.read_csv(csvFilePath)

    ### Texto para agregar en latex
    latex_line_1 = "\center\n" # Centering Table
    latex_line_2 = "\hline \multicolumn{4}{|c|}{Datos obtenidos de programa} \\\\\n\n" # Pie de tabla

    ### Generando contenido de tabla
    latex_table = (
        data[['Item',"Value", "Partial", "Total"]]
        .rename(columns={
            "Item": "Intem",
            "Value": "Valor",
            "Partial": "Tiempo Partial",
            "Total": "Tiempo acumulado"
        })
        .to_latex(
            index=False,
            float_format=custom_format,
            caption="Datos correspondientes al archivo programa"+str(programNum)+".csv luego de 25 segundos",
            label="tab:programa"+str(programNum),
            escape=False,
            column_format="|c|c|c|c|"
        )
    )

```

```
    ### Editando contenido de tabla para mejor visibilidad
    latex_lines = latex_table.split('\n')
    latex_lines.insert(1, latex_line_1)
    latex_lines[0] = "\\begin{table}[ht!]"
    latex_lines.insert(-4, latex_line_2)
    latex_table = '\n'.join(latex_lines)

    ### Creando archivo latex
    with open(latexFilePath, "w") as Tfile:
        Tfile.write(latex_table)


# Data de archivos CSV
## Archivo1
fileName = 'infoproceso1'
path_experient = os.path.join("../../", "build", fileName+".csv")
path_latex = os.path.join("../", "Latex/src/tables/"+str(fileName)+".tex") # Archivo
CreateLatexTable(path_experient, path_latex, 1)

## Archivo2
fileName = 'infoproceso2'
path_experient = os.path.join("../../", "build", fileName+".csv")
path_latex = os.path.join("../", "Latex/src/tables/"+str(fileName)+".tex") # Archivo
CreateLatexTable(path_experient, path_latex, 2)

## Archivo3
fileName = 'infoproceso3'
path_experient = os.path.join("../../", "build", fileName+".csv")
path_latex = os.path.join("../", "Latex/src/tables/"+str(fileName)+".tex") # Archivo
CreateLatexTable(path_experient, path_latex, 3)
```

Referencias

Academy, K. (s.f.). *Generar números aleatorios*. Consultado el 29 de septiembre de 2024, desde https://es.khanacademy.org/computing/ap-computer-science-principles/x2d2f703b37b450a3:simulations/x2d2f703b37b450a3:simulating-randomness/a/generating-random-numbers

de Cabezón, E. S. (2022). *El azar es imposible (al menos en los ordenadores) — El drama de LOS NÚMEROS ALEATORIOS*. Consultado el 29 de septiembre de 2024, desde https://www.youtube.com/watch?v=RzEjqJHW-NU

IONOS. (2023). *Comando ps de Linux: para comprobar los procesos en ejecución*. Consultado el 29 de septiembre de 2024, desde https://www.ionos.com/es-us/digitalguide/servidores/configuracion/comando-ps-de-linux/

Linux. (2023). *killall(1) — Linux manual page*. Consultado el 30 de septiembre de 2024, desde https://man7.org/linux/man-pages/man1/killall.1.html

Tanenbaum, A. S., & Woodhull, A. S. (1997). *Sistemas Operativos: Diseño e implementación* (R. Escalona, Trad.; Segunda edición). Prentice Hall.