2023

Memoria

PRÁCTICA 3

PAULA CASTILLEJO BRAVO

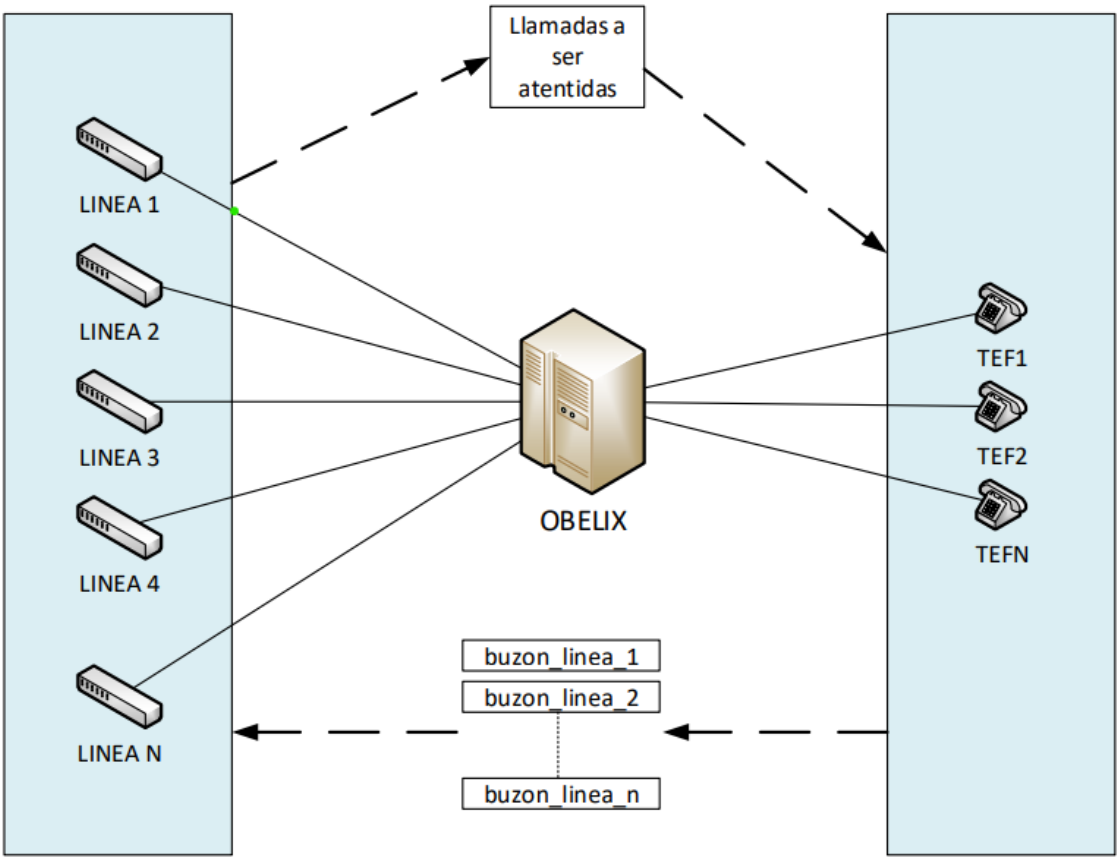
ÍNDICE:

1. Especificación

Estamos muy emocionados, y seguro que vosotros también, porque hemos descubierto la implementación de paso de mensajes entre procesos. Esto nos permite seguir trabajando sobre nuestro maravilloso sistema Obelix.

En este sistema, nos encontraremos con un proceso Manager que será el encargado del control de Obelix. Tendremos, por un lado, líneas que recibirán llamadas telefónicas, y, por otro lado, tendremos teléfonos que serán los encargados de atender estas llamadas.

Este sería el esquema de Obelix para paso de mensajes.



Construya un sistema compuesto por tres ejecutables que simule el funcionamiento que se detalla a continuación, y que es, la especificación básica de funcionamiento del sistema Obelix.

- Manager: Deberá cumplir las siguientes especificaciones:
  - Inicializar las colas de mensajes necesarias para establecer la comunicación entre líneas y teléfonos, así como, la comunicación entre los teléfonos y cada una de las líneas.
  - Lanzará el número de procesos teléfono y procesos línea definidos en este caso como constantes del sistema.
  - El sistema, gracias al paso de mensajes, no tendrá finalización, por lo que el Apagado Controlado de Obelix se realizará mediante la pulsación de Ctrl-C.
  - Forzará la finalización de todos los procesos línea actualmente en funcionamiento.
  - Forzará la finalización de todos los procesos teléfono existentes.
  - Por último, pero no menos importante, liberará todos los recursos utilizados. (tablas de procesos, colas de mensajes, etc.)

- Teléfono: Realizará las siguientes tareas dentro de su función:
  - Se inician en espera de una llamada, informando de ello: "Teléfono[UID] en espera…".
  - Cuando reciben una llamada, tienen una conversación variable entre 10..20 segundos e informan de ello: "Teléfono[UID] en conversación de llamada desde la línea: [buzon_linea_n]…".
  - Cuando finalicen su conversación, notificarán a la línea este suceso y mostrarán un mensaje: Teléfono [%d] ha colgado la llamada. [buzon_linea_n].
  - Volverán a su estado inicial de espera de llamada.
  - *En ningún caso finalizarán su ejecución salvo cuando el proceso Manager lo considere oportuno.*
- Línea: Realizará las siguientes tareas dentro de su función:
  - Recibirá como parámetro su cola de mensajes para el proceso de notificación de teléfonos a líneas (buzon_linea_n).
  - Se inician y esperan una llamada (Simulado con un rand() entre 1..30 segundos), informando de ello: "Linea[UID] esperando llamada".
  - Cuando reciben la llamada (Finalización de rand()):
    - Enviará un mensaje a la cola de llamadas para ser atendidas.
    - Esperarán la notificación desde un teléfono, de que la llamada ha finalizado.
    - Notificarán la recepción del teléfono y volverán al estado de espera de llamada.
  - En ningún caso finalizarán su ejecución salvo cuando el proceso Manager lo considere oportuno.

2. Compilación

La fórmula para compilar un archivo en lenguaje C, conociendo de antemano que se trabaja en una terminal en el sistema operativo linux, consiste en el comando gcc -I include/ -o manager src/manager.c, en donde la opción -I indica los directorios donde han de buscarse bibliotecas mientras –o compila un programa en C y genera un archivo ejecutable.

3. Ejecución

Una vez poseyendo el ejecutable del programa, la ejecución de este es un paso más simple desde terminal, ubicándonos en el directorio donde está el ejecutable y escribiendo, en el siguiente orden, ./manager . No es necesario ningún paso de parámetros puesto que dentro del mismo código utilizamos variables definidas con las cifras[1] concretas para su correcta ejecución.

[1] El sistema operativo tiene un máximo de 10 mensajes para las colas de mensajes, por lo que el número de líneas de Obelix será de 10.

2.1. Ejemplo de ejecución

paula_2002@paula-VivoBook-ASUSLaptop-X415EA-F415EA:~/UCLM/LaboratorioPCTR/pctr_p3_src/esqueleto$ ./exec/manager

[MANAGER] 3 telefonos creados

Teléfono [5482] en espera...

Teléfono [5481] en espera...

Teléfono [5483] en espera...

Linea [5484] esperando llamada...

[MANAGER] 10 lineas creados

Linea [5485] esperando llamada...

Linea [5487] esperando llamada...

Linea [5489] esperando llamada...

Linea [5486] esperando llamada...

Linea [5491] esperando llamada...

Linea [5488] esperando llamada...

Linea [5493] esperando llamada...

Linea [5490] esperando llamada...

Linea [5492] esperando llamada...

Linea [5491] recibida llamada (/buzon_linea_7)

Linea [5491] esperando fin de conversación...

Teléfono [5482] en conversacion de llamada desde Linea: /buzon_linea_7

Linea [5492] recibida llamada (/buzon_linea_8)

Linea [5492] esperando fin de conversación...

Teléfono [5481] en conversacion de llamada desde Linea: /buzon_linea_8

Linea [5487] recibida llamada (/buzon_linea_3)

Linea [5487] esperando fin de conversación...

Teléfono [5483] en conversacion de llamada desde Linea: /buzon_linea_3

Linea [5486] recibida llamada (/buzon_linea_2)

Linea [5486] esperando fin de conversación...

Linea [5490] recibida llamada (/buzon_linea_6)

Linea [5490] esperando fin de conversación...

Linea [5488] recibida llamada (/buzon_linea_4)

Linea [5488] esperando fin de conversación...

Linea [5485] recibida llamada (/buzon_linea_1)

Linea [5485] esperando fin de conversación...

Linea [5489] recibida llamada (/buzon_linea_5)

Linea [5489] esperando fin de conversación...

Linea [5493] recibida llamada (/buzon_linea_9)

Linea [5493] esperando fin de conversación...

Teléfono [5481] ha colgado la llamada: /buzon_linea_8

Teléfono [5481] en espera...

Teléfono [5481] en conversacion de llamada desde Linea: /buzon_linea_2

Linea [5492] conversación finalizada...

Linea [5492] esperando llamada...

Teléfono [5482] ha colgado la llamada: /buzon_linea_7

Teléfono [5482] en espera...

Teléfono [5482] en conversacion de llamada desde Linea: /buzon_linea_6

Linea [5491] conversación finalizada...

Linea [5491] esperando llamada...

Teléfono [5483] ha colgado la llamada: /buzon_linea_3

Teléfono [5483] en espera...

Teléfono [5483] en conversacion de llamada desde Linea: /buzon_linea_4

Linea [5487] conversación finalizada...

Linea [5487] esperando llamada...

Linea [5484] recibida llamada (/buzon_linea_0)

Linea [5484] esperando fin de conversación...

Linea [5491] recibida llamada (/buzon_linea_7)

Linea [5491] esperando fin de conversación...

Teléfono [5481] ha colgado la llamada: /buzon_linea_2

Teléfono [5481] en espera...

Teléfono [5481] en conversacion de llamada desde Linea: /buzon_linea_1

Linea [5486] conversación finalizada...

Linea [5486] esperando llamada...

Teléfono [5482] ha colgado la llamada: /buzon_linea_6

Teléfono [5482] en espera...

Teléfono [5482] en conversacion de llamada desde Linea: /buzon_linea_5

Linea [5490] conversación finalizada...

Linea [5490] esperando llamada...

Teléfono [5483] ha colgado la llamada: /buzon_linea_4

Teléfono [5483] en espera...

Teléfono [5483] en conversacion de llamada desde Linea: /buzon_linea_9

Linea [5488] conversación finalizada...

Linea [5488] esperando llamada...

Linea [5488] recibida llamada (/buzon_linea_4)

Linea [5488] esperando fin de conversación...

Teléfono [5481] ha colgado la llamada: /buzon_linea_1

Teléfono [5481] en espera...

Teléfono [5481] en conversacion de llamada desde Linea: /buzon_linea_0

Linea [5485] conversación finalizada...

Linea [5485] esperando llamada...

Linea [5492] recibida llamada (/buzon_linea_8)

Linea [5492] esperando fin de conversación...

Linea [5485] recibida llamada (/buzon_linea_1)

Linea [5485] esperando fin de conversación...

Linea [5487] recibida llamada (/buzon_linea_3)

Linea [5487] esperando fin de conversación...

Linea [5486] recibida llamada (/buzon_linea_2)

Linea [5486] esperando fin de conversación...

Teléfono [5482] ha colgado la llamada: /buzon_linea_5

Teléfono [5482] en espera...

Teléfono [5482] en conversacion de llamada desde Linea: /buzon_linea_7

Linea [5489] conversación finalizada...

Linea [5489] esperando llamada...

Linea [5490] recibida llamada (/buzon_linea_6)

Linea [5490] esperando fin de conversación...

Teléfono [5483] ha colgado la llamada: /buzon_linea_9

Teléfono [5483] en espera...

Teléfono [5483] en conversacion de llamada desde Linea: /buzon_linea_4

Linea [5493] conversación finalizada...

Linea [5493] esperando llamada...

Teléfono [5481] ha colgado la llamada: /buzon_linea_0

Teléfono [5481] en espera...

Teléfono [5481] en conversacion de llamada desde Linea: /buzon_linea_8

Linea [5484] conversación finalizada...

Linea [5484] esperando llamada...

Linea [5484] recibida llamada (/buzon_linea_0)

Linea [5484] esperando fin de conversación...

Linea [5489] recibida llamada (/buzon_linea_5)

^C

[Linea 5493] Finalizado (SIGINT)

[Linea 5484] Finalizado (SIGINT)

[Telefono 5481] Finalizado (SIGINT)

[MANAGER] Terminacion del programa (Ctrl + C).

[Linea 5492] Finalizado (SIGINT)

----- [MANAGER] Terminar con los procesos hijos ejecutándose -----

[MANAGER] Terminando proceso LINEA [5484]...

[MANAGER] Terminando proceso LINEA [5485]...

[MANAGER] Terminando proceso LINEA [5486]...

[MANAGER] Terminando proceso LINEA [5487]...

[MANAGER] Terminando proceso LINEA [5488]...

[MANAGER] Terminando proceso LINEA [5489]...

[MANAGER] Terminando proceso LINEA [5490]...

[MANAGER] Terminando proceso LINEA [5491]...

[MANAGER] Terminando proceso LINEA [5492]...

[MANAGER] Terminando proceso LINEA [5493]...

[Linea 5488] Finalizado (SIGINT)

[Linea 5487] Finalizado (SIGINT)

[Linea 5489] Finalizado (SIGINT)

[Linea 5491] Finalizado (SIGINT)

[Linea 5486] Finalizado (SIGINT)

[Linea 5490] Finalizado (SIGINT)

[Linea 5485] Finalizado (SIGINT)

[Telefono 5483] Finalizado (SIGINT)

[Telefono 5482] Finalizado (SIGINT)

----- [MANAGER] Terminar con los procesos hijos ejecutándose -----

[MANAGER] Terminando proceso TELEFONO [5481]...

[MANAGER] Terminando proceso TELEFONO [5482]...

[MANAGER] Terminando proceso TELEFONO [5483]...

3. Discusión

Una vez se domina los procedimientos de compilación y ejecución del programa, se proporcionará constancia acerca de las conclusiones[2] durante todo el proyecto abarcado en las siguientes tres semanas.

3.1. MakeFile

En primera instancia, un archivo makefile es un conjunto de herramientas make para construir y crear ejecutables y objetos en base de los archivos dependientes .c y .h, ubicados en los directorios /src e /include respectivamente. Los nombres lógicos principales que lo componen serían:

```
1    DIROBJ := obj/
2    DIREXE := exec/
3    DIRHEA := include/
```

[2] Al tratarse este una práctica realizada de forma individual, no se ha generado ninguna discusión o debate como tal, pero sí existe la ayuda por parte de otros compañeros de clase y del profesorado de la asignatura en cuestión.

```
4       DIRSRC := src/

5       CFLAGS := -I$(DIRHEA) -c -Wall -ggdb
6       LDFLAGS := -lpthread -lrt
7       CC := gcc
```

Desde la línea 1 hasta la 4 se instancian los directorios en su correspondiente variable, en las líneas 5 y 6 indicamos las banderas Wall, la cual reportará errores y advertencias, y lpthread, dónde se pretende cargar las bibliotecas (existentes o inventadas), y en la línea 7 indicamos el compilador. Ahora trataremos con las reglas implícitas en el documento:

```
8       all : dirs manager telefono linea

9       dirs:
10              mkdir -p $(DIROBJ) $(DIREXE)

11      manager: $(DIROBJ)manager.o
12              $(CC) -o $(DIREXE)$@ $^ $(LDFLAGS)

13      telefono: $(DIROBJ)telefono.o
14              $(CC) -o $(DIREXE)$@ $^ $(LDFLAGS)

15      linea: $(DIROBJ)linea.o
16              $(CC) -o $(DIREXE)$@ $^ $(LDFLAGS)

17      $(DIROBJ)%.o: $(DIRSRC)%.c
18              $(CC) $(CFLAGS) $^ -o $@

20      clean :
21              rm -rf *~ core $(DIROBJ) $(DIREXE) $(DIRHEA)*~ $(DIRSRC)*~
```

En la línea 8, all se usa para enumerar todos los subobjetivos necesarios para construir el proyecto. En las líneas 11, 13 y 15 se forman los ejecutables, siendo manager el del proceso principal. En la línea 17 se procede a establecer los ficheros objetos. En la línea 20, la función clean sirve para limpiar todas las reglas anteriores.

3.2. Manager.c

Redactar los once métodos, sin contar el main, del programa no es moco de pavo, por eso se detallará cada uno en orden de llamada dentro del archivo.

3.2.1. main()

Como es bien sabido por los programadores a estas alturas, este método es el punto de entrada de un programa ejecutable; esto es, dónde se inicia y finaliza el control del programa. Aquí se llama a los procedimientos indispensables para su conveniente funcionalidad: crear_buzones(), instalar_manejador_senhal(), iniciar_tabla_procesos(), crear_procesos() y esperar_procesos().

```
35      int main(int argc, char *argv[])
36      {
37              // Creamos los buzones
38              crear_buzones();
39
40              // Manejador de Ctrl-C
41              instalar_manejador_senhal();
42
43              // Crea Tabla para almacenar los pids de los procesos
44              iniciar_tabla_procesos(NUMTELEFONOS,NUMLINEAS);
45
46              // Tenemos todo
47              // Lanzamos los procesos
48              crear_procesos(NUMTELEFONOS,NUMLINEAS);
49
50              // Esperamos a que finalicen las lineas
51              esperar_procesos();
52
53              return EXIT_SUCCESS;
54      }
```

3.2.2. Crear_buzones()

Esta función nueva (en contraste con la práctica anterior) fabrica los buzones con el comando mq_open() que más adelante intercambiarán el flujo de mensajes para similar las llamadas telefónicas. Mq_open opera con el nombre de la cola de mensajes, la lectura/creación de la cola, los permisos del propietario y el atributo con la información para trabajar (máximo número de mensajes y tamaño estándar de los mensajes: 10 y 64, respectivamente). La cadena caux se utiliza como auxiliar para componer la cantidad de buzones para las líneas en correspondencia al número de líneas creadas, 3, con ayuda del bucle. Por último, se realizan comprobaciones acerca de si se han creado bien los buzones, notificando por pantalla en caso de error.

```
211     void crear_buzones(){
212             struct mq_attr mqAttr;
213             char caux[30];
214
215             mqAttr.mq_maxmsg = NUMLINEAS;
```

```
216             mqAttr.mq_msgsize = TAMANO_MENSAJES;
217
218             qHandlerLlamadas = mq_open(BUZON_LLAMADAS, O_WRONLY | O_CREAT,
S_IWUSR | S_IRUSR, &mqAttr);
219
220             if (qHandlerLlamadas == -1){
221                     fprintf(stderr, "\tError creando HandlerLlamadas: %s\n", strerror(errno));
222             }
223
224             int i;
225
226             mqAttr.mq_maxmsg = 1;
227
228             for (i = 0; i < NUMLINEAS; i++) {
229                     sprintf(caux, "%s%d", BUZON_LINEAS, i);
230                     qHandlerLineas[i] = mq_open(caux, O_WRONLY | O_CREAT, S_IWUSR |
S_IRUSR, &mqAttr);
231
232                     if (qHandlerLineas[i] == -1){
233                             fprintf(stderr, "\tError creando HandlerLinea %s: %s\n", caux,
strerror(errno));
234                     }
235             }
236     }
```

3.2.3. instalar_manejador_senhal()

Instalar un manejador de señales resulta eficiente al momento de querer frenar el programa de manera manual desde la terminal, pero se debe tener en cuenta que se necesita un controlador auxiliar, manejador_senal(), para identificar el tipo de señal y no cortar toda la operación. Sin no se procesa la señal, el sistema forzará un cierre sin preguntarse que acción se le ha indicado. Nuestro proyecto pretende detectar señales de terminación de programa mediante la pulsación de teclas Ctrl + C.

```
56      void instalar_manejador_senhal()
57      {
58              if (signal(SIGINT, manejador_senal) == SIG_ERR)
59              {
60                      fprintf(stderr, "\t[MANAGER] Error al instalar el manejador de senhal:
%s.\n", strerror(errno));
61                      exit(EXIT_FAILURE);
62              }
63      }
```

3.2.4. manejador_senal()

El controlador simplemente llama a las funciones de terminar_procesos() y liberar_recursos(), realizando un próspero cierre del ejecutable.

```
65	void manejador_senal(int sign)
66	{
67		printf("\n[MANAGER] Terminacion del programa (Ctrl + C).\n");
68		terminar_procesos();
69		liberar_recursos();
70		exit(EXIT_SUCCESS);
71	}
```

3.2.5. terminar_procesos()

Primero, se terminar los procesos correspondientes a las líneas y después a los teléfonos. Para ahorrar tiempo y recursos, se procede a llamara al método terminar_procesos_especificos() con parámetros ajustados en parejas de tabla de procesos y cantidad de procesos.

```
73	void terminar_procesos()
74	{
75		terminar_procesos_especificos(g_process_lineas_table, g_lineas_Processes);
76		terminar_procesos_especificos(g_process_telefonos_table, g_telefonos_Processes);
77	}
```

3.2.4. terminar_procesos_específicos()

Esta función finalizará los procesos hijos en ejecución de forma generalizada al recibir los datos por entrada, corroborando si los identificadores almacenados en las tablas de procesos y lanzando una llamada al sistema y matando todos los hijos ipso facto.

```
79	void terminar_procesos_especificos(struct TProcess_t *process_table, int process_num)
80	{
81		int i;
82		printf("\n----- [MANAGER] Terminar con los procesos hijos ejecutándose ----- \n");
83
84		for (i = 0; i < process_num; i++)
85		{
86			if (process_table[i].pid != 0)
87			{
88				printf("[MANAGER] Terminando proceso %s [%d]...\n",
process_table[i].clase, process_table[i].pid);
89				if (kill(process_table[i].pid, SIGINT) == -1)
90				{
```

```
91                                  fprintf(stderr, "[MANAGER] Error al usar kill() en proceso %d: %s.\n", process_table[i].pid, strerror(errno));
92                              }
93                          }
94                  }
95          }
```

3.2.7. liberar_recursos()

Empleando las funciones free(), mq_close() y mq_unlink() nuestro sistema elimina la memoria preciamente reservada por un malloc(), cierra el descriptor de cola de mensajes (una vez con el buzón de las llamadas y diez con cada buzón de las líneas) y elimina la cola de mensajes; en orden de mención.

```
127     void liberar_recursos()
128     {
129             int i; char caux[30];
130
131             free(g_process_telefonos_table);
132             free(g_process_lineas_table);
133
134             mq_close(qHandlerLlamadas);
135             mq_unlink(BUZON_LLAMADAS);
136
137             for (i = 0; i < NUMLINEAS; i++) {
138                     sprintf(caux, "%s%d", BUZON_LINEAS, i);
139                     mq_close(qHandlerLineas[i]);
140                     mq_unlink(caux);
141             }
142     }
```

3.2.5. iniciar_tabla_procesos()

A la hora de iniciar las tablas de los procesos, al inicio se realizaba la suma de los procesos de líneas y teléfonos cuando estos en realidad se emplean por separado. Por tanto, intuimos que habrá dos bucles en donde los pid de cada una de las tablas de los procesos se inicializará a 0.

```
108     void iniciar_tabla_procesos(int n_processes_telefono, int n_processes_linea)
109     {
110             int i;
111             g_lineas_Processes = n_processes_linea, g_telefonos_Processes = n_processes_telefono;
112
```

```
113             g_process_lineas_table = malloc(g_lineas_Processes * sizeof(struct TProcess_t));
114             g_process_telefonos_table = malloc(g_telefonos_Processes * sizeof(struct
TProcess_t));
115
116             for (i = 0; i < g_lineas_Processes; i++)
117             {
118                     g_process_lineas_table[i].pid = 0;
119             }
120
121             for (i = 0; i < g_telefonos_Processes; i++)
122             {
123                     g_process_telefonos_table[i].pid = 0;
124             }
125     }
```

3.2.6. crear_procesos()

La funcionalidad aquí se distingue de un simple vistazo. Cada bucle se encarga de llamar a las funciones lanzar_proceso_linea() y lanzar_proceso_telefono(), respectivamente[3], el número de veces necesaria.

```
144     void crear_procesos(int numTelefonos, int numLineas)
145     {
146             int i;
147
148             for (i = 0; i < numLineas; i++)
149             {
150                     lanzar_proceso_linea(i);
151             }
152             printf("\t[MANAGER] %d lineas creados\n", numLineas);
153
154             for (i = 0; i < numTelefonos; i++)
155             {
156                     lanzar_proceso_telefono(i);
157             }
158             printf("\t[MANAGER] %d telefonos creados\n", numTelefonos);
159             sleep(1);
```

[3] Crear antes los procesos de línea que los de teléfono. Si entra una llamada se quedará almacenada en la cola de “Llamadas a ser atendidas” hasta que estén disponibles los teléfonos.

```
160     }
```

3.2.7. lanzar_proceso_linea()

La primitiva fork() . Mediante el switch controlamos el resultado de la llamada al sistema: si sale -1 significa que hubo un error, terminando los procesos y liberando los recursos; si se produce un cero continuamos con exec(), lanzando el proceso en cuestión con el susodicho nombre de la cola de mensaje como parámetro. Al final, se registra en la tabla de procesos el pid junto al tipo de clase.

```
162     void lanzar_proceso_linea(const int indice_tabla)
163     {
164             pid_t pid;
165             char caux[30];
166
167             sprintf(caux, "%s%d", BUZON_LINEAS, indice_tabla);
168
169             switch (pid = fork())
170             {
171             case -1:
172                     fprintf(stderr, "\t[MANAGER] Error al lanzar proceso lineas: %s.\n",
strerror(errno));
173                     terminar_procesos();
174                     liberar_recursos();
175                     exit(EXIT_FAILURE);
176             case 0:
177                     if (execl(RUTA_LINEA, CLASE_LINEA, caux, NULL) == -1)
178                     {
179                             fprintf(stderr, "\t[MANAGER] Error usando execl() en el proceso %s:
%s.\n", CLASE_LINEA, strerror(errno));
180                             exit(EXIT_FAILURE);
181                     }
182             }
183
184             g_process_lineas_table[indice_tabla].pid = pid;
185             g_process_lineas_table[indice_tabla].clase = CLASE_LINEA;
186     }
```

3.2.8. lanzar_proceso_telefono()

Este apartado repite los mismo pasos que el anterior, salvo por la entrada de parámetros, la cual no se produce.

```
188     void lanzar_proceso_telefono(const int indice_tabla)
189     {
```

```
190             pid_t pid;
191
192             switch (pid = fork())
193             {
194             case -1:
195                     fprintf(stderr, "\t[MANAGER] Error al lanzar proceso telefono: %s.\n",
strerror(errno));
196                     terminar_procesos();
197                     liberar_recursos();
198                     exit(EXIT_FAILURE);
199             case 0:
200                     if (execl(RUTA_TELEFONO, CLASE_TELEFONO, NULL) == -1)
201                     {
202                             fprintf(stderr, "\t[MANAGER] Error usando execl() en el poceso %s:
%s.\n", CLASE_TELEFONO, strerror(errno));
203                             exit(EXIT_FAILURE);
204                     }
205             }
206
207             g_process_telefonos_table[indice_tabla].pid = pid;
208             g_process_telefonos_table[indice_tabla].clase = CLASE_TELEFONO;
209     }
```

3.2.9. esperar_procesos()

En contraste, cuando esperamos por los procesos, no se procede a realizar nada puesto que, si todo el programa se ejecuta a la perfección, nunca se abandonaran los procesos hijos línea.c y teléfono.c a menos que se produzca una interrupción externa. Está opción ha sido óptima puesto que si es necesario esperar “por algo” para que el propio manager.c no se cierre al crear los procesos hijos.

```
97      void esperar_procesos()
98      {
99              int i;
100
101             for (i = 0; i < NUMLINEAS; i++)
102             {
103                     waitpid(g_process_lineas_table[i].pid, 0, 0);
104             }
105     }
```

3.3. Telefono.c

Reducido al empleo de un solo método sin parámetros, el archivo telefono.c se ocupa de la espera, recepción y finalización de una llamada. Sin embargo, se detallarán los pasos vinculados con el flujo de mensajes mediante colas de mensajes entre un teléfono y la llamada atendida.

Hay que tener en cuenta, sin lugar a duda, la paralización de las líneas una vez el teléfono detecta la primera (tengamos en cuenta que se ocupará un teléfono por línea según estas se registran en el buzón de llamadas): problema del bloqueo. Después, se ingresará a la sección de funcionalidad del módulo, abriremos el buzón destinado de las llamadas (línea 38) y recibiremos el nombre de la línea que realiza la llamada a través del buzón recién abierto (línea 51). Más tarde, con la finalización de la llamada, abrimos el buzón de la línea la cual acababa de efectuar la llamada y avisamos a esta que se ha colgado.

Esto se exhibe en el siguiente fragmento de código:

```
21  void telefono(){
22      // Define variables locales
23      int pid = getpid(), rc = 0;
24      mqd_t qHandlerLlamadas;
25      mqd_t qHandlerLinea;
26      char buzonLinea[TAMANO_MENSAJES];
27      char buffer[TAMANO_MENSAJES+1];
28
29      srand(pid);
30
31      // Retrollamada de finalización.
32      if (signal(SIGINT, controlador) == SIG_ERR) {
33          fprintf(stderr, "Abrupt termination.\n");
34          exit(EXIT_FAILURE);
35      }
36
37      //Recuperar buzones
38      qHandlerLlamadas = mq_open(BUZON_LLAMADAS, O_RDWR);
39
40      if (qHandlerLlamadas == -1){
41          fprintf(stderr, "\tError abriendo HandlerLlamadas: %s\n", strerror(errno));
42      }
43
44      // Se pone en estado de libre incrementando el número de teléfonos libres
45      while(1){
```

```
46
47                      // Mensaje de Espera
48                      printf("Teléfono [%d] en espera...\n", pid);
49
50                      //Flujo de mensajes entrada a telefono
51                      rc = mq_receive(qHandlerLlamadas, (char *) &buzonLinea, sizeof(buzonLinea), NULL);
52                      sprintf(buffer,"%s",buzonLinea);
53
54                      if (rc == -1){
55                              fprintf(stderr, "\tError recibiendo mensaje en HandlerLlamadas: %s\n", strerror(errno));
56                      }
57
58                      // Mensaje en conversacion
59                      printf("Teléfono [%d] en conversacion de llamada desde Linea: %s\n", pid, buffer);
60
61                      // Espera en conversación
62                      sleep(rand() % 10 + 10);
63
64                      //Fin de conversación
65                      printf("Teléfono [%d] ha colgado la llamada: %s\n", pid, buffer);
66
67                      //Flujo de mensajes salida a linea
68                      qHandlerLinea = mq_open(buzonLinea, O_RDWR);
69
70                      if (qHandlerLinea == -1){
71                              fprintf(stderr, "\tError abriendo HandlerLinea: %s\n", strerror(errno));
72                      }
73
74                      mq_send(qHandlerLinea, buzonLinea, sizeof(buzonLinea), 0);
75                      close(qHandlerLinea);
76              }
77      }
```

3.4. Líneas.c

Para concluir, se hablará del archivo linea.c. En este proceso se inicia una llamada, la cual espera hasta ser recibida por un teléfono. El método linea es el único que añade valor en este archivo.

El flujo de mensaje se podría describir como inverso al del apartado anterior, coordinado con la entrada y salida de información. Es necesario verificar el adecuado paso de argumentos, alertando en caso negativo y cerrando el proyecto. Aquí podemos abrir las dos colas de mensajes desde el principio, continuado con una comprobación de que se hizo una apertura idónea (también hay comprobaciones en los envíos y recibos). Luego de llamar a la semilla aleatoria, ingresamos en un bucle infinito donde se repetirá el mismo procedimiento de actuación: un mq_send() al buzón de llamadas con el nombre de la línea que necesita ser atendida notificará a teléfono cuando iniciar una conversación y un mq_receive() avisará que se colgó el teléfono; con varios mensajes por pantalla sobre las esperar de las llamadas o la finalización de la misma.

```
22  void lineas(int argc, char *num_linea){
23      // Define variables locales
24      int pid = getpid(), rc = 0;
25      mqd_t qHandlerLlamadas;
26      mqd_t qHandlerLinea;
27      char buzonLinea[TAMANO_MENSAJES];
28      char buffer[TAMANO_MENSAJES+1];
29
30      // Retrollamada de finalización.
31      if (signal(SIGINT, controlador) == SIG_ERR) {
32          fprintf(stderr, "Abrupt termination.\n");
33          exit(EXIT_FAILURE);
34      }
35
36      // Verifica los parámetros
37      if (argc != 2)
38      {
39          fprintf(stderr, "Error. Usa: ./exec/linea <cola_linea_llamante>.\n");
40          exit(EXIT_FAILURE);
41      }
42      sprintf(buzonLinea,"%s",num_linea);
43
44      //Recuperar buzones
45      qHandlerLlamadas = mq_open(BUZON_LLAMADAS, O_RDWR);
46      qHandlerLinea = mq_open(buzonLinea, O_RDWR);
```

```
47
48              if (qHandlerLlamadas == -1){
49                      fprintf(stderr, "\tError abriendo HandlerLlamadas: %s\n", strerror(errno));
50              }
51
52              if (qHandlerLinea == -1){
53                      fprintf(stderr, "\tError abriendo HandlerLinea: %s\n", strerror(errno));
54              }
55
56              // Inicia Random
57              srand(pid);
58
59              while(1){
60                      // Realiza una espera entre 1..60 segundos
61                      printf("Linea [%d] esperando llamada...\n", pid);
62                      sleep(rand() % 30 + 1);
63
64                      //Flujo de mensajes salida a telefono
65                      mq_send(qHandlerLlamadas, buzonLinea, sizeof(buzonLinea), 0);
66
67                      // Espera recibir llamada
68                      printf("Linea [%d] recibida llamada (%s)\n", pid, buzonLinea);
69
70                      //Espera fin conversación
71                      printf("Linea [%d] esperando fin de conversación...\n", pid);
72
73                      //Flujo de mensajes entrada a linea
74                      rc = mq_receive(qHandlerLinea, (char *) &buffer, sizeof(buffer), NULL);
75
76                      if(rc == -1){
77                              fprintf(stderr, "\tError recibiendo mensaje en HandlerLinea: %s\n",
strerror(errno));
78                      }
79
80                      // Llamada finalizada
81                      printf("Linea [%d] conversación finalizada...\n", pid);
82              }
83      }
```

4. Bibliografía

(s.f.). Recuperado el 05 de 04 de 2023, de Manual de Linux en Línea: https://man7.org/linux/man-pages/index.html

Vallejo, D., González, C., & Albusac, J. (2015). *Programación Concurrente y Tiempo Real* (Tercera ed.). Ciudad Real. Recuperado el 04 de 05 de 2023