Anexo B

Filósofos comensales

B.1. Enunciado

Los filósofos se encuentran comiendo o pensando. Todos comparten una mesa redonda con cinco sillas, una para cada filósofo. En el centro de la mesa hay una fuente de arroz y en la mesa sólo hay cinco palillos, de manera que cada filósofo tiene un palillo a su izquierda y otro a su derecha.

Cuando un filósofo piensa, entonces se abstrae del mundo y no se relaciona con ningún otro filósofo. Cuando tiene hambre, entonces intenta acceder a los palillos que tiene a su izquierda y a su derecha (necesita ambos). Naturalmente, un filósofo no puede quitarle un palillo a otro filósofo y sólo puede comer cuando ha cogido los dos palillos. Cuando un filósofo termina de comer, deja los palillos y se pone a pensar.

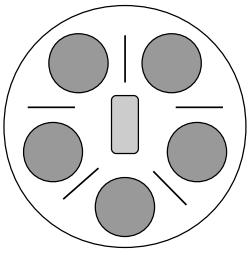

Figura B.1: Esquema gráfico del problema original de los filósofos comensales (*dining philosophers*).

B.2. Código fuente

Listado B.1: Makefile para la compilación automática

```
DIROBJ := obj/
DIREXE := exec/
DIRSRC := src/

CFLAGS := -c -Wall
LDLIBS := -lrt
CC := gcc

all : dirs manager filosofo

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

manager: $(DIROBJ)manager.o
    $(CC) -o $(DIREXE)$@ $^ $(LDLIBS)

filosofo: $(DIROBJ)filosofo.o
    $(CC) -o $(DIREXE)$@ $^ $(LDLIBS)

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

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

Listado B.2: Archivo manager.c

```c
#include <mqueue.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <wait.h>

#define BUZON_MESA    "/buzon_mesa"
#define BUZON_PALILLO "/buzon_palillo_" /* Offset! */
#define FILOSOFOS 5

pid_t pids[FILOSOFOS];
mqd_t qHandlerMesa;
mqd_t qHandlerPalillos[FILOSOFOS];

void controlador (int senhal);
void liberarecursos ();
void finalizarprocesos ();

int main (int argc, char *argv[]) {
  int i;
  struct mq_attr mqAttr;
  char buffer[64], caux[30], filosofo[1];
  char buzon_palillo_izq[30], buzon_palillo_der[30];

  // Reseteo del vector de pids.
  memset (pids, 0, sizeof(pid_t) * (FILOSOFOS));

  // Atributos del buzón mesa.
  mqAttr.mq_maxmsg = (FILOSOFOS - 1); mqAttr.mq_msgsize = 64;

  // Retrollamada de finalización.
  if (signal(SIGINT, controlador) == SIG_ERR) {
    fprintf(stderr, "Abrupt termination.\n");
    exit(EXIT_FAILURE);
  }

  // Creación de buzones...
  qHandlerMesa = mq_open(BUZON_MESA, O_WRONLY| O_CREAT, S_IWUSR | S_IRUSR, &mqAttr);
  for (i = 0; i < (FILOSOFOS - 1); i++)
    // Para evitar el interbloqueo...
    // Sólo 4 filósofos (máximo) intentan acceder a los palillos.
    mq_send(qHandlerMesa, buffer, sizeof(buffer), 1);

  // Para los buzones de los palillos.
  mqAttr.mq_maxmsg = 1;

  // Un buzón por palillo...
  for (i = 0; i < FILOSOFOS; i++) {
    sprintf(caux, "%s%d", BUZON_PALILLO, i);
    qHandlerPalillos[i] = mq_open(caux, O_WRONLY | O_CREAT, S_IWUSR | S_IRUSR, &mqAttr);
    // Palillo inicialmente libre.
    mq_send(qHandlerPalillos[i], buffer, sizeof(buffer), 0);
  }
```

```
58
59    // Lanzamiento de procesos filósofo.
60    for (i = 0; i < FILOSOFOS ; i++)
61      if ((pids[i] = fork()) == 0) {
62        sprintf(filosofo, "%d", i);
63        sprintf (buzon_palillo_izq, "%s%d", BUZON_PALILLO, i);
64        sprintf (buzon_palillo_der, "%s%d", BUZON_PALILLO, (i + 1) % FILOSOFOS);
65        execl("./exec/filosofo", "filosofo", filosofo, BUZON_MESA,
66              buzon_palillo_izq, buzon_palillo_der, NULL);
67      }
68
69    for (i = 0; i < FILOSOFOS; i++) waitpid(pids[i], 0, 0);
70    finalizarprocesos(); liberarecursos();
71    printf ("\n Fin del programa\n");
72    return 0;
73  }
74
75  void controlador (int senhal) {
76    finalizarprocesos(); liberarecursos();
77    printf ("\n Fin del programa (Control + C)\n");
78    exit(EXIT_SUCCESS);
79  }
80
81  void liberarecursos () {
82    int i; char caux[30];
83
84    printf ("\n Liberando buzones... \n");
85    mq_close(qHandlerMesa); mq_unlink(BUZON_MESA);
86
87    for (i = 0; i < FILOSOFOS; i++) {
88      sprintf (caux, "%s%d", BUZON_PALILLO, i);
89      mq_close(qHandlerPalillos[i]); mq_unlink(caux);
90    }
91  }
92
93  void finalizarprocesos () {
94    int i;
95    printf ("-------------- Terminando ------------- \n");
96    for (i = 0; i < FILOSOFOS; i++) {
97      if (pids[i]) {
98        printf ("Finalizando proceso [%d]... ", pids[i]);
99        kill(pids[i], SIGINT); printf ("<Ok>\n");
100     }
101   }
102 }
```

Listado B.3: Archivo filosofo.c

```c
#include <mqueue.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define MAX_TIME_PENSAR 7
#define MAX_TIME_COMER 5

void filosofo (char *filosofo, char *buzon_mesa,
               char *buzon_palillo_izq, char *buzon_palillo_der);
void controlador (int senhal);

int main (int argc, char *argv[]) {
  filosofo(argv[1], argv[2], argv[3], argv[4]);
  return 0;
}

void filosofo (char *filosofo, char *buzon_mesa,
               char *buzon_palillo_izq, char *buzon_palillo_der) {
  mqd_t qHandlerMesa, qHandlerIzq, qHandlerDer;
  int n_filosofo;
  char buffer[64];

  // Retrollamada de finalización.
  if (signal(SIGINT, controlador) == SIG_ERR) {
    fprintf(stderr, "Abrupt termination.\n"); exit(EXIT_FAILURE);
  }
  n_filosofo = atoi(filosofo);

  // Recupera buzones.
  qHandlerMesa = mq_open(buzon_mesa, O_RDWR);
  qHandlerIzq = mq_open(buzon_palillo_izq, O_RDWR);
  qHandlerDer = mq_open(buzon_palillo_der, O_RDWR);

  srand((int)getpid());
  while (1) {
    printf("[Filosofo %d] pensando...\n", n_filosofo);
    sleep(rand() % MAX_TIME_PENSAR); // Pensar.

    mq_receive(qHandlerMesa, buffer, sizeof(buffer), NULL);

    // Hambriento (coger palillos)...
    mq_receive(qHandlerIzq, buffer, sizeof(buffer), NULL);
    mq_receive(qHandlerDer, buffer, sizeof(buffer), NULL);
    // Comiendo...
    printf("[Filosofo %d] comiendo...\n", n_filosofo);
    sleep(rand() % MAX_TIME_COMER); // Comer.
    // Dejar palillos...
    mq_send(qHandlerIzq, buffer, sizeof(buffer), 0);
    mq_send(qHandlerDer, buffer, sizeof(buffer), 0);

    mq_send(qHandlerMesa, buffer, sizeof(buffer), 0);
  }
}


```

```
58 void controlador (int senhal) {
59   printf("[Filosofo %d] Finalizado (SIGINT)\n", getpid());
60   exit(EXIT_SUCCESS);
61 }
```