Examen de Sistemas Operativos (Parte práctica). Primera Convocatoria. Curso 2008/09. Tercero Ingeniería Técnica Informática	
Nombre:	☐ Gestión ☐ Sistemas
Apellidos:	*5 de Febrero de 2009*

[Cuestión 1.] Resuelva con una línea de comandos UNIX las siguientes tareas:

Apartado a.- Obtenga un listado de todos los ficheros regulares del sistema terminados en *".y"* ó *".c"*, con permiso de lectura para el grupo.

```
$ls -lR / | grep "\.[yc]$"| grep "^-...r"
```

Apartado b.- Copie los ficheros que terminan en *".h"* que se encuentran en el directorio temporal del sistema, al directorio padre de su directorio actual de trabajo.

```
$cp /tmp/*.h ..
```

Apartado c.- Cambie convenientemente los permisos del fichero *"bol4-ej1.c"* del subdirectorio "*fuentes*" de su directorio *HOME* atendiendo a las siguientes indicaciones. Sólo el propietario y el grupo puedan leerlo. Sólo el propietario debe poder modificarlo. Ningún usuario podrá ejecutarlo.

```
$chmod 640 $HOME/fuentes/bol4-ej1.c
```

Apartado d.- Compile y enlace los archivos de código fuente en lenguaje C "*principal.c"* y "*funciones.c"*. Ambos se encuentran en su directorio actual de trabajo. Debe enlazarlos con el fichero "*semaph.o*" que se encuentra en el directorio *"/usr/lib"*. Como resultado se deberá generar un archivo ejecutable cuyo nombre será "*ejercicio1*", situado en el directorio *"/usr/bin*".

```
$gcc principal.c funciones.c /usr/lib/semaph.o -o /usr/bin/ejercicio1
```

Puntuación: 2 ptos. ***Tiempo estimado: 10 min.***

Examen de Sistemas Operativos (Parte práctica). Primera Convocatoria. Curso 2008/09. Tercero Ingeniería Técnica Informática	
Nombre:	☐ Gestión ☐ Sistemas
Apellidos:	*5 de Febrero de 2009*

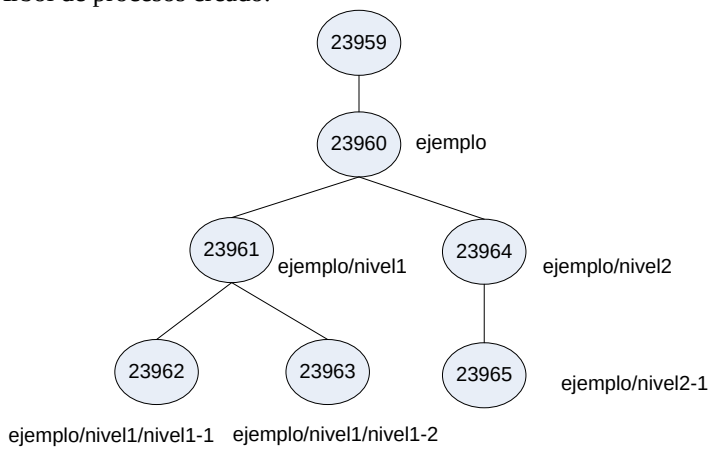
[Cuestión 2.] Realice un programa en C, "*recorrido_multiproceso*", que recorra de forma recursiva un determinado directorio y muestre por pantalla la ruta de todos los ficheros regulares que éste contiene. La búsqueda en cada uno de los directorios (directorio principal y subdirectorios) deberá realizarse en un proceso distinto para optimizar la búsqueda. La ruta del directorio a recorrer se recibirá como primer argumento.

La siguiente figura muestra un ejemplo del programa en ejecución.

Estructura directorios:

```
+ejemplo
+--->nivel1
      +--->nivel1-1
      +--->nivel1-2
            +--->fichero4
            +--->fichero5
      +--->fichero2
      +--->fichero3
+--->nivel2
      +--->nivel2-1
            +--->fichero6
+--->fichero1
```

Árbol de procesos creado:




```
murillo>recorrido_multiproceso ./ejemplo
Proceso hijo 23963: ejemplo/nivel1/nivel1-2/fichero4
Proceso hijo 23963: ejemplo/nivel1/nivel1-2/fichero5
Proceso hijo 23961: ejemplo/nivel1/fichero2
Proceso hijo 23961: ejemplo/nivel1/fichero3
Proceso hijo 23965: ejemplo/nivel2/nivel2-1/fichero6
Proceso hijo 23960: ejemplo/fichero1
```

NOTAS

- No deje procesos huérfanos ni zombies.
- No se permite el uso de la llamada system.

Puntuación: 4 ptos. ***Tiempo estimado: 30 min.***

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#define LONG 256

int imprimir_directorio(char *ruta);

int main(int argc, char *argv[])
{

      if (argc!=2)
            fprintf(stderr,"Uso: %s <dir>\n", argv[0]);
```

```
      /* Imprimir el directorio */
      if (imprimir_directorio(argv[1]) == -1)
      {
            fprintf(stderr,"Ocurrio un error al imprimir el directorio\n");
            return EXIT_FAILURE;
      }

      return 0;
}

int imprimir_directorio(char *ruta)
{
       struct dirent *d;
       struct stat buf;
       DIR *dir;
       char archivo[LONG];
       int i,status;
       pid_t childpid;

       childpid=fork();

       if (childpid==-1)
       {
            perror("Error al crear el proceso");
            return -1;
       }

       if (childpid == 0)
       {
            /* Abrir directorio */
            if ((dir=opendir(ruta))==NULL)
            {
                  fprintf(stderr, "Error al abrir el directorio: %s\n", ruta);
                  return -1;
            }

            while ((d=readdir(dir))!=NULL)
            {
                  if (strcmp(d->d_name,".")!=0 && strcmp(d->d_name, "..")!=0)
                  {
                        /* Obtenemos la ruta del archivo actual */
                        sprintf(archivo,"%s/%s",ruta,d->d_name);

                        if (stat(archivo, &buf)==-1)
                        {
                              fprintf(stderr,  "Error  al  hacer  stat  sobre  %s\n",
archivo);
                              return -1;
                        }

                        if (buf.st_mode & S_IFREG)
                              printf("Proceso %ld: %s \n", (long)getpid(), archivo);

                        if (buf.st_mode & S_IFDIR)
                              imprimir_directorio(archivo);
                  }
            }
            /* Cerrar directorio */
            closedir(dir);
            exit(EXIT_SUCCESS);
      }else
      {
            while(childpid!=wait(&status));
      }
            return 0;
}
```

Examen de Sistemas Operativos (Parte práctica). Primera Convocatoria. Curso 2008/09. Tercero Ingeniería Técnica Informática	
Nombre:	☐ Gestión ☐ Sistemas
Apellidos:	*5 de Febrero de 2009*

[Cuestión 3.] Se desea implantar una bolera altamente automatizada. Dos de las piezas fundamentales de la misma serán el dispensador de tickets y el dispensador de zapatos. Cuando un cliente llegue a la bolera, deberá solicitar un ticket en el dispensador oportuno y espera a que éste se lo dé. Una vez el cliente tenga el ticket, solicitará unos zapatos y esperará a que éstos sean dispensados. Finalmente, el cliente podrá hacer uso de la bolera. Se pide desarrollar tres programas en C para simular el comportamiento sincronizado de clientes, dispensador de tickets y dispensador de zapatos. Considere para ello las siguientes restricciones de sincronización:

- Todas las esperas serán no ocupadas.
- Ambos dispensadores permanecerán a la espera mientras no haya peticiones por atender.
- Dos clientes no pueden acceder a la vez al dispensador de tickets.
- Suponga la bolera inicialmente vacía.

Suponga implementadas las siguientes funciones:

- void dispensar-ticket(void): Simula la dispensación de un ticket.
- void dispensar-zapatos(void): Simula la dispensación de unos zapatos.
- void recoger-ticket(void): Simula la recogida del ticket por parte de un cliente.
- void recoger-zapatos(void): Simula la recogida de los zapatos por parte de un cliente.
- void jugar(void): Simula una partida de bolos.

NOTAS:

- Emplee la biblioteca de semáforos estudiada en clase.
- Asegúrese de eliminar los semáforos al recibir la señal SIGTERM

Puntuación: 4 ptos. ***Tiempo estimado: 40 min.***

iniciador.h

```
Semaph dispensar_ticket;
Semaph dispensar_zapatos;
Semaph recoger_ticket;
Semaph recoger_zapatos;
Semaph excmut;

void iniciador(void);
void terminador(int);
```

iniciador.c

```
void iniciador(void)
{

   dispensar_ticket = semaph_create("dispensar_ticket", 0);
   recoger_ticket = semaph_create("recoger_ticket",0);
   dispensar_zapatos = semaph_create("dispensar_zapatos",0);
   recoger_zapatos = semaph_create("recoger_zapatos",0);
   excmut = semaph_create("excmut",1);

   if (dispensar_ticket == -1 || recoger_ticket == -1 || dispensar_zapatos == -1
|| recoger_zapatos == -1 || excmut == -1)
   {
         fprintf(stderr, "Error al crear los semáforos\n");
         exit(EXIT_FAILURE);
   }
```

```
   }

   void terminador(int sig)
   {
      printf("Destruyendo semáforos...\n");
      semaph_destroy("dispensar_ticket",dispensar_ticket);
      semaph_destroy("dispensar_zapatos",dispensar_zapatos);
      semaph_destroy("recoger_ticket",recoger_ticket);
      semaph_destroy("recoger_zapatos",recoger_zapatos);
      semaph_destroy("excmut",excmut);

      exit(0);
}
```

disp_tickets.c

```
   void dispensarTicket(void);

   int main(void)
   {

      // Manejador para la señal SIGTERM
      if (signal (SIGTERM, terminador) == SIG_ERR)
      {
            perror("Error en signal");
            exit(EXIT_FAILURE);
      }

      iniciador();

      while (1)
      {
            semaph_down(dispensar_ticket);
            dispensarTicket();
            semaph_up(recoger_ticket);
      }
   }

   void dispensarTicket(void)
   {
      sleep(1);
      printf("Ticket listo\n");
}
```

disp_zapatos.c

```
   void dispensarZapatos(void);

   int main(void)
   {
      iniciador();

      // Manejador para la señal SIGTERM
      if (signal (SIGTERM, terminador) == SIG_ERR)
      {
            perror("Error en signal");
            exit(EXIT_FAILURE);
      }


      while (1)
      {
            semaph_down(dispensar_zapatos);
            dispensarZapatos();
            semaph_up(recoger_zapatos);
```

```
      }
   }

   void dispensarZapatos(void)
   {
      sleep(2);
      printf("Zapatos listos\n");
}
```

cliente.c

```
   void recogerTicket(void);
   void recogerZapatos(void);
   void jugar(void);

   int main(void)
   {
      iniciador();

      // Manejador para la señal SIGTERM
      if (signal (SIGTERM, terminador) == SIG_ERR)
      {
            perror("Error en signal");
            exit(EXIT_FAILURE);
      }


      while (1)
      {
            // Ticket
            semaph_up(dispensar_ticket);
            semaph_down(recoger_ticket);
            semaph_down(excmut);
            recogerTicket();
            semaph_up(excmut);

            // Zapatos
            semaph_up(dispensar_zapatos);
            semaph_down(recoger_zapatos);
            recogerZapatos();


            // Jugar
            jugar();
      }
   }

   void recogerTicket(void)
   {
      printf("Cliente: Ya tengo el ticket!!\n");
   }

   void recogerZapatos(void)
   {
      printf("Cliente: Ya tengo los zapatos\n");
```

```
   }

   void jugar(void)
   {
      printf("Que partidazo!\n");
}
```