\* Para la realización del presente examen se dispondrá de **3 horas, incluido el test.  
\* No** se pueden utilizar libros **ni** apuntes. Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen

**Ejercicio 2 (2 puntos)**

En un computador que utiliza política de planificación cíclica, se ejecutan los siguientes procesos de los que se conoce su tiempo de llegada y su tiempo total de ejecución:

|  |  |  |
| --- | --- | --- |
| Proceso | Tiempo de llegada | Duración |
| A | 0 | 400 |
| B | 125 | 200 |
| C | 150 | 400 |
| D | 175 | 300 |

A) Rellene una tabla como la siguiente, asumiendo que la rodaja de tiempo es de 100 ms., indicando qué proceso está en ejecución y el estado de la cola de procesos listos en cada instante de tiempo. Indique en la columna eventos en qué momento del tiempo se produce para cada proceso, su llegada, su inicio y su finalización.

|  |  |  |  |
| --- | --- | --- | --- |
| Instante | En ejecución | Cola de listos | Eventos |
|  |  |  |  |
| … | … | … |  |

B) Rellene una tabla como la siguiente, indicando para cada proceso los tiempos de llegada, servicio, inicio, fin, retorno, espera y espera normalizado.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Proceso | Llegada | Servicio | Inicio | Fin | Retorno | Espera | Retorno  Normalizado |
| A | 0 |  |  |  |  |  |  |
| B | 125 |  |  |  |  |  |  |
| C | 150 |  |  |  |  |  |  |
| D | 175 |  |  |  |  |  |  |

C) Repita el apartado A, considerando un rodaja de 200 ms.

|  |  |  |  |
| --- | --- | --- | --- |
| Instante | En ejecución | Cola de listos | Eventos |
|  |  |  |  |
| … | … | … |  |

D) Repita el apartado B, para el caso de una rodaja de tiempo de 200 ms.

SOLUCIÓN

A)

|  |  |  |  |
| --- | --- | --- | --- |
| Instante | En ejecución | Cola de listos | Eventos |
| 0 | A(400) |  | Llegada A, Inicio A |
| 100 | A(300) |  |  |
| 125 | A(275) | B(200) | Llegada B |
| 150 | A(250) | B(200), C(400) | Llegada C |
| 175 | A(225) | B(200), C(400), D(300) | Llegada D |
| 200 | B(200) | C(400), D(300), A(200) | Inicio B |
| 300 | C(400) | D(300), A(200), B(100) | Inicio C |
| 400 | D(300) | A(200), B(100), C(300) | Inicio D |
| 500 | A(200) | B(100), C(300), D(200) |  |
| 600 | B(100) | C(300), D(200), A(100) |  |
| 700 | C(300) | D(200), A(100) | Fin B |
| 800 | D(200) | A(100), C(200) |  |
| 900 | A(100) | C(200), D(100) |  |
| 1000 | C(200) | D(100) | Fin A |
| 1100 | D(100) | C(100) |  |
| 1200 | C(100) |  | Fin D |
| 1300 |  |  | Fin C |

B)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Proceso | Llegada | Servicio | Inicio | Fin | Retorno | Espera | Retorno  Normalizado |
| A | 0 | 400 | 0 | 1000 | 1000 | 600 | 1000/400 |
| B | 150 | 200 | 200 | 700 | 550 | 350 | 550/200 |
| C | 150 | 400 | 300 | 1300 | 1150 | 750 | 1150/400 |
| D | 175 | 300 | 400 | 1200 | 1025 | 725 | 1025/300 |

C)

|  |  |  |  |
| --- | --- | --- | --- |
| Instante | En ejecución | Cola de listos | Eventos |
| 0 | A(400) |  | Llegada A, Inicio A |
| 125 | A(275) | B(200) | Llegada B |
| 150 | A(250) | B(200), C(400) | Llegada C |
| 175 | A(225) | B(200), C(400), D(300) | Llegada D |
| 200 | B(200) | C(400), D(300), A(200) | Inicio B |
| 400 | C(400) | D(300), A(200) | Inicio C, Fin B |
| 600 | D(300) | A(200), C(200) | Inicio D |
| 800 | A(200) | C(200), D(100) |  |
| 1000 | C(200) | D(100) | Fin A |
| 1200 | D(100) |  | Fin C |
| 1300 |  |  | Fin D |

D)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Proceso | Llegada | Servicio | Inicio | Fin | Retorno | Espera | Retorno  Normalizado |
| A | 0 | 400 | 0 | 1000 | 1000 | 600 | 1000/400 |
| B | 150 | 200 | 200 | 400 | 250 | 50 | 50/200 |
| C | 150 | 400 | 400 | 1200 | 1050 | 650 | 650/400 |
| D | 175 | 300 | 600 | 1300 | 1125 | 825 | 825/300 |

**Ejercicio 3 (2 puntos)**

Realizar un programa que **cree 12 threads** que simularán la entrada de 3 personas en una sala. En la **sala solo puede haber o 0 o 3** **personas** por lo que los threads deberán **esperar** para entrar a **que haya un grupo de 3**. Estarán **esperando** en la sala durante un tiempo aleatorio **entre 1 y 4** segundos y después **saldrán de la sala las 3 personas** a la vez. **Hasta que no salgan las 3 no podrán entrar otras 3** en la sala. *Nota: use la función srandom para la espera aleatoria.*

Cada vez que se cree un thread, este debe escribir el mensaje “**Thread creado: ID**”. Cuando estén dentro de la sala escribirán en pantalla el mensaje “**Persona dentro: ID**”. Cuando salgan escribirán el mensaje “**Persona fuera: ID**”. ”, ID serán número entre 0 y 11 asignado al crear el thread.

La aplicación se deberá realizar **utilizando** como **mecanismo de sincronización** entre los threads **únicamente** **mutex** y **variables condicionales**.

Ejemplo de ejecución:

Thread creado : 0

Thread creado : 1

Thread creado : 2

Persona dentro : 2

Persona dentro : 0

Persona dentro : 1

Thread creado : 3

Thread creado : 4

Thread creado : 5

Thread creado : 6

Thread creado : 7

Thread creado : 8

Thread creado : 9

Thread creado : 10

Thread creado : 11

Persona fuera : 1

Persona fuera : 2

Persona fuera : 0

Persona dentro : 5

Persona dentro : 3

Persona dentro : 4

Persona fuera : 4

Persona fuera : 5

Persona fuera : 3

Etc.

**Solución**

pthread\_attr\_t attr;

pthread\_t idth[12];

pthread\_mutex\_t mtx;

pthread\_cond\_t varcond;

int yacopiada=0;

int personasEntrada=0;

pthread\_mutex\_t mtxEntrada;

pthread\_cond\_t varcondEntrada;

int lleno=0;

int personasSalida=0;

pthread\_mutex\_t mtxSalida;

pthread\_cond\_t varcondSalida;

void \*hilo(void \*num) {

int id,espera;

srandom(time (NULL));

//Copio el parámetro recibido en una variable local en una zona protegida por

//mutex para tener acceso exclusivo

//Después señalizo que ha finalizado la operación para que se entere el main

pthread\_mutex\_lock (&mtx);

id=\*((int \*)num);

yacopiada=1;

pthread\_cond\_signal(&varcond);

pthread\_mutex\_unlock (&mtx);

printf("\tThread id creado : %d\n",id);

//Cola de espera para entrar, se van apuntando en personasEntrada;

pthread\_mutex\_lock (&mtxEntrada);

while (lleno==1 )

pthread\_cond\_wait(&varcondEntrada, &mtxEntrada);

personasEntrada++;

if (lleno==0 && personasEntrada==3){

lleno=1;

pthread\_cond\_broadcast(&varcondEntrada);

}

while ( personasEntrada <3)

pthread\_cond\_wait(&varcondEntrada, &mtxEntrada);

pthread\_mutex\_unlock (&mtxEntrada);

printf("\tPersona id dentro : %d\n",id);

espera=random()%4 +1;

sleep(espera);

pthread\_mutex\_lock (&mtxEntrada);

personasEntrada--; //Se prepara para salir

pthread\_mutex\_unlock (&mtxEntrada);

pthread\_mutex\_lock (&mtxSalida);

personasSalida++;

if (personasSalida==3)

pthread\_cond\_broadcast(&varcondSalida);

if (personasSalida <3)

pthread\_cond\_wait(&varcondSalida, &mtxSalida);

printf("\tPersona id fuera : %d\n",id);

personasSalida--;

if (personasSalida==0){ //Permito entrar a otras 3

lleno=0;

pthread\_cond\_broadcast(&varcondEntrada);

}

pthread\_mutex\_unlock (&mtxSalida);

pthread\_exit(0);

}

int main(){

int i;

pthread\_mutex\_init (&mtx, NULL);

pthread\_attr\_init(&attr);

pthread\_mutex\_init (&mtxEntrada, NULL);

for (i=0; i<12; i++) {

pthread\_create(&idth[i],&attr,hilo,&i);

//Espero a que el thread creado utilice el valor de i

pthread\_mutex\_lock (&mtx);

while (yacopiada==0)

pthread\_cond\_wait(&varcond, &mtx);

yacopiada=0;

pthread\_mutex\_unlock (&mtx);

}

// Espero la finalización del thread

for (i=0; i<10; i++)

pthread\_join(idth[i],NULL);

return(0);

}

**Ejercicio 4 (2 puntos)**

El siguiente programa en C (incompleto y simplificado) pretende implementar la configuración de procesos de la figura y el correspondiente flujo de datos con **tuberías**. El proceso A (padre) tomará datos del archivo “*entrada.txt*” que enviará al proceso hijo B; este, a su vez, le enviará los datos a su proceso “hermano”, hijo C, que se encargará de devolverlos al padre para que los imprima por pantalla. La ejecución de los procesos concluye cuando el proceso A (padre) recibe la cadena "exit".



Lea el código e incluya las **definiciones y llamadas pertinentes en C** en las zonas señaladas para implementar el comportamiento descrito en el texto.

|  |  |
| --- | --- |
| void hijoB()  {  char buf[4096];  int exit\_flag = 0;    **... // OCHO**  do {  read(...); **// NUEVE**  if (strncmp("exit\n", buf, strlen("exit\n")) == 0)  exit\_flag = 1;    write(...); **// DIEZ**  } while (!exit\_flag);  **... // ONCE**  } | **int ...; // UNO**  int main(void) {  char buf[4096];  int pid, i;  int exit\_flag = 0;  **... // DOS**  for (i=0; i < 2; i++) {  **... // TRES**  if (pid == 0) {  switch(i) {  case 0: /\* tratamiento hijo B. \*/  hijoB();  exit(EXIT\_SUCCESS);  case 1: /\* tratamiento hijo C. \*/  hijoC();  exit(EXIT\_SUCCESS);  }  } else if (pid > 0 && i == 1) { /\* Tratamiento Padre \*/    **... // CUATRO**  do {  read(STDIN\_FILENO, buf, sizeof(buf));    if (strncmp("exit\n", buf, strlen("exit\n")) == 0)  exit\_flag = 1;  write(...); **// CINCO**    read(...); **// SEIS**    write(STDOUT\_FILENO, buf, strlen(buf));  } while (!exit\_flag);  }  }  **... // SIETE**    do {  pid = wait(NULL);  } while (pid > 0);  } /\* Fin del Main \*/ |
| void hijoC()  {  char buf[4096];  int exit\_flag = 0;  **... // DOCE**    do {  read(...); **// TRECE**    if (strncmp("exit\n", buf, strlen("exit\n")) == 0)  exit\_flag = 1;    write(...); **// CATORCE**    } while (!exit\_flag);  **... // QUINCE**  } |

**Solución**

1. int fds\_pipe1[2], fds\_pipe2[2], fds\_pipe3[2]; // UNO
2. void hijoB()
3. {
4. char buf[4096];
5. int exit\_flag = 0;
7. close(fds\_pipe1[1]); // OCHO
8. close(fds\_pipe2[0]);
10. do {
11. read(fds\_pipe1[0], buf, sizeof(buf)); // NUEVE
12. if (strncmp("exit\n", buf, strlen("exit\n")) == 0)
13. exit\_flag = 1;
15. write(fds\_pipe2[1], buf, strlen(buf)); // DIEZ
16. } while ( !exit\_flag);
17. close(fds\_pipe1[0]); // ONCE
18. close(fds\_pipe2[1]);
19. }
20. void hijoC()
21. {
22. char buf[4096];
23. int exit\_flag = 0;
24. close(fds\_pipe2[1]); // DOCE
25. close(fds\_pipe3[0]);
26. do {
27. numbytes = read(fds\_pipe2[0], buf, sizeof(buf)); // TRECE
29. if (strncmp("exit\n", buf, strlen("exit\n")) == 0)
30. exit\_flag = 1;
32. write(fds\_pipe3[1], buf, strlen(buf)); // CATORCE
34. } while ( !exit\_flag);
35. close(fds\_pipe3[1]); // QUINCE
36. close(fds\_pipe2[0]);
37. }
39. int main(void) {
40. char buf[4096];
41. int pid, i;
42. int exit\_flag = 0;
43. int fd;
44. pipe(fds\_pipe1); // DOS
45. pipe(fds\_pipe2);
46. pipe(fds\_pipe3);
47. for (i=0; i<2; i++) {
48. pid = fork(); // TRES
49. if (pid == 0) {
50. switch(i) {
51. case 0:
52. /\* tratamiento hijo B. \*/
53. hijoB();
54. exit(EXIT\_SUCCESS);
55. case 1:
57. /\* tratamiento hijo C. \*/
58. hijoC();
59. exit(EXIT\_SUCCESS);
60. }
61. } else if (pid > 0 && i == 1) {
62. /\* Tratamiento del Padre \*/
63. close(fds\_pipe1[0]); // CUATRO
64. close(fds\_pipe3[1]);
66. fd = open (“entrada.txt”, O\_RDONLY);
67. If ( fd == -1 ) {
68. printf (“error al abriir archivo entrada \n”);
69. exit();
70. }
71. do {
73. i = read(fd, buf, sizeof(buf));
75. // se sale si llega exit o archivo vacío
76. if (I == 0) || (strncmp("exit\n", buf, strlen("exit\n")) == 0)
77. exit\_flag = 1;
78. write(fds\_pipe1[1], buf, strlen(buf)); // CINCO
80. read(fds\_pipe3[0], buf, sizeof(buf)); // SEIS
82. write(STDOUT\_FILENO, buf, strlen(buf));
83. } while (numbytes > 0 && !exit\_flag);
84. } else if (ret == -1) {
85. perror("fallo en fork");
86. exit(EXIT\_FAILURE);
87. }
88. }
89. // El padre cierra la tubería antes de esperar y salir
90. close(fds\_pipe1[1]); // SIETE
91. close(fds\_pipe3[0]);
93. do {
94. pid = wait(NULL);
95. } while (pid > 0);
96. }

**Ejercicio 5 ( 2 puntos).**

Escriba un programa *ls*. Dicho programa toma como parámetro el nombre del directorio de entrada e imprime por pantalla una lista de archivos en ese directorio, uno por línea, de forma similar al comando “*ls -1*”. La impresión de fichero no seguirá ningún orden concreto. Si el directorio no existe dará un error y termina. La función devolverá -1 en caso de error, y 0 si todo funciona correctamente.

Adicionalmente, se desea implementar un programa *cat*. Este programa recibe como argumento de entrada un *nombre de fichero*, comprueba si existe e imprime su contenido por pantalla. La función devolverá -1 en caso de error, y 0 si todo funciona correctamente.

Se pide:

1. Escriba el pseudocódigo de la función *ls* definida arriba, de forma que se cumpla la interfaz y funcionamiento descritos.
2. Desarrolle, en lenguaje C, el pseudocódigo del apartado anterior.
3. Escriba el pseudocódigo de la función *cat* definida arriba, de forma que se cumpla la interfaz y funcionamiento descritos.
4. Desarrolle, en lenguaje C, el pseudocódigo del apartado anterior.
5. Si queremos compilar el programa *ls.c* y obtener como salida el ejecutable*ls*. ¿Qué comando debemos usar?
6. Escriba un comando en Linux que: liste las entradas dentro del directorio actual , filtre si existe el archivo “kk.c” con el comando grep y se lo pase al al comando cat para que lo muestre por pantalla.

**Solución: (explicación: la parte azul es obligatoria, la parte roja puede considerarse opcional)**

**a)**

|  |
| --- |
| **Programa ls**  Abrir directorio\_entrada y comprobar error de apertura. |
| Devolver -1 si procede. |
| Mientras haya ficheros en el directorio: |
| Leer entrada de directorio. |
| Imprimir el nombre del fichero |
|  |
|  |
|  |

**b)**

**Programa ls**

|  |
| --- |
| #include <sys/types.h> // opendir, readdir  #include <dirent.h> // opendir, readdir  #include <stdlib.h> // NULL  #include <unistd.h> // write  #include <string.h> // strlen  int main (int argc, char \*argv[]) {  struct dirent \*entrada;  DIR \*fd1;    fd1 = opendir(argv[1]);  if(fd1 == NULL) {  return -1;  }  int c, r1, r2;  do{  entrada = readdir(fd1);  if(entrada != NULL) {  write(STDOUT\_FILENO, entrada->d\_name, strlen(entrada->d\_name));  write(STDOUT\_FILENO,"\n",2);  // printf(“%s\n”,entrada->d\_name); // faltaría #include <stdio.h>  }  }while(entrada != NULL);  closedir(fd1);  return 0;  } |

**c)**

|  |
| --- |
| Abrir fichero\_entrada y comprobar error de apertura. |
| Devolver -1 si procede. |
| Mientras queden caracteres en fichero\_entrada: |
| Leer carácter de fichero\_entrada. |
| Escribir carácter en salida estándar. |
| Comprobar lectura/escritura.  Cerrar fichero y devolver -1 si procede. |
| Cerrar fichero. |
| Devolver 0. |

**d)**

#include <sys/types.h> // open

#include <sys/stat.h> // open

#include <fcntl.h> // open

#include <unistd.h> //STDOUT\_FILENO

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

int fd1 = open(argv[1], O\_RDONLY);

if(fd1 < 0) {

return -1;

}

char c;

int r1, r2;

do{

r1 = read(fd1, &c, 1);

if(r1 < 0) {

close(fd1);

return -1;

}

if(r1 != 0) {

r2 = write(STDOUT\_FILENO, &c, 1);

if(r2 < 0) {

close(fd1);

return -1;

}

}

}while(r1 > 0);

close(fd1);

return 0;

}

**e)**

gcc ls.c -o ls

**f)**

**cat < ‘**ls /. | grep “kk.c”’