

Sistemas Operativos

Departamento de Computación - FCEyN - UBA

Primer cuatrimestre de 2025

Primer recuperatorio - 1er cuatrimestre de 2025

Nombre y apellido: Pedro Fuenle, Uffely

Nº orden: 44 L.U.: 108122 Cant. hojas: 4

1	2	3	4	Nota
25	25	25	25	100

(A)

- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido, LU y número de orden. Hojas sin identificar no serán corregidas. Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas. Entregue esta hoja junto al examen, la misma no se incluye en la cantidad total de hojas entregadas. La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen dos notas: I (Insuficiente): 0 a 64 pts y A (Aprobado): 65 a 100 pts.

Ejercicio 1.(25 puntos)

Se desea implementar un Servidor para gestionar las actas de finales de la Universidad, utilizando sockets. Cada docente se conecta al Servidor para crear el acta de su materia (un sólo docente por materia), indicando un código numérico (int) que representa a la materia, y comienza a cargar notas. Por cada estudiante que rinde el final, el docente envía un mensaje indicando el número de libreta universitaria (strings con la forma NNNN/NN) y la nota final (int). Al finalizar la carga, el Servidor debe cerrar el acta y cerrar la conexión. El Servidor debe ser capaz de manejar simultáneamente N docentes cargando al mismo tiempo notas finales. Se dispone de una estructura llamada Acta, y de las siguientes funciones:

```
Acta crearActa(int codMateria); //Crea un acta para la materia con el código indicado
void agregarNota(Acta acta, char* lu, int nota); //Agrega nota de estud con lu al acta
void cerrarActa(Acta acta); //Cierra el acta
```

Completar el código del proceso Servidor y de uno de los procesos Docente.

```
int main_servidor() {
    int servidor_fd, cliente_fd;
    struct sockaddr_un direccion;
    socklen_t addrlen = sizeof(direccion);
    direccion.sin_family = AF_UNIX;
    strcpy(direccion.sun_path, "unix_socket");
    unlink(direccion.sun_path);
    servidor_fd = socket(AF_UNIX, SOCK_STREAM, 0);
    ...
}

int main_docente() {
    struct sockaddr_un servidor;
    servidor.sun_family = AF_UNIX;
    strcpy(servidor.sun_path, "unix_socket");
    int sock = socket(AF_UNIX, SOCK_STREAM, 0);
    ...
}
```

Ejercicio 2.(25 puntos)

Una ciudad inteligente tiene sensores de calidad del aire distribuidos en distintos barrios, que reportan concentraciones de gases (CO_2 , NO_2 , etc.). Un proceso asociado a cada sensor de la ciudad recopila y preprocesa periódicamente sus datos. Luego, en el centro de monitoreo, un proceso central consolida los datos de los sensores de la ciudad y genera información que se envía a un servidor web donde se mantiene un mapa de contaminación en tiempo real.

Escribir el pseudocódigo que modele este escenario de forma tal que esté libre de deadlock y condiciones de carrera y considerando que:

- Los procesos asociados a sensores conocen la función preprocDataSensor() que levanta y preprocesa datos del sensor.

- El proceso central conoce la función `consolidarDataSensores()` que procesa los datos previamente trabajados por los demás procesos, y envía el resultado al servidor web.
- Para optimizar el uso de recursos, el proceso central debe esperar a tener N resultados de los procesos de los sensores antes de consolidarlos para el mapa.
- Por limitaciones de recursos, se pueden ejecutar sólo hasta M ($M < N$) `preprocDataSensor()` en simultáneo.

Ejercicio 3.(25 puntos)

Se tiene un servidor en la nube que ejecuta múltiples procesos pertenecientes a distintos usuarios. Cada usuario puede lanzar varios procesos que compiten por el uso de CPU.

El scheduler de este sistema asigna CPU entre usuarios, no entre procesos individuales. Cada usuario se trata como un grupo de procesos, y dentro de cada grupo se establece una política de scheduling interna. Además, existen k categorías de usuarios que determinan su prioridad, donde 0 es el nivel más alto (usuarios Premium) y $k-1$ es el nivel más bajo.

El sistema garantiza que los usuarios usen la CPU de manera justa y evita que un usuario obtenga más procesamiento simplemente lanzando más procesos. El scheduler tiene acceso a métricas que indica el porcentaje de uso de CPU por parte de un usuario en el tiempo reciente.

Responder por verdadero o falso y justificar en ambos casos.

- Todos los usuarios (que son grupos de procesos) deben organizarse en una única cola con política RR con quantum q . Durante el turno de cada usuario, se ejecutarán sus procesos siguiendo su política interna. Cuando termine su turno, se desalojará y pasará al siguiente.
- Todos los usuarios (que son grupos de procesos) deben separarse en colas de prioridad. Para elegir al siguiente usuario dentro de una prioridad, se usa un esquema FIFO.
- Los usuarios de prioridad baja podrían nunca acceder a utilizar el CPU.

Ejercicio 4.(25 puntos)

Considerar el siguiente programa en C:

```
int X[N];
int i = 0;
int a;
int b;
// ...

while (i < N) {
    a = X[i];
    b = X[N - 1 - i];
    X[i] = X[i] + f(a, b);
    i += 3;
}
```

Sabemos que cada página de memoria puede contener exactamente P enteros, la memoria física tiene F marcos de página y sólo cada lectura o escritura a $X[i]$ implica un acceso a memoria.

- Dado $N = 12$, $P = 2$, listar la secuencia de números de página accedidos y justificar cómo se derivan esos números de página a partir de los valores de i . Considerar que el código, los valores de N e i , y la definición de $f()$ se encuentran en una página cargada que no puede desalojarse, y que $X[0]$ se encuentra al inicio de la página 1.
- Usando $F = 3$ marcos de página y la secuencia de pedidos calculada en el inciso anterior, simular los algoritmos de reemplazo LRU y Second Chance. Para cada uno, indicar la cantidad total de fallos de página y el contenido de los marcos tras cada acceso.

1) Empleo haciendo algunas observaciones

. Asumo que en el cliente tiene un arreglo (arreglo-data) de k posiciones. En cada una hay un struct de este tipo

```
TYPEDEF STRUCT DATA-T {
    CHAR[8] LU;
    INT NOTA;
} DATA-T;
```

Este struct conocido (redefinido) por el cliente y el servidor

. El cliente (cliente) conoce también su código de materia, en una variable de tipo int llamada const.

) Como no recuerdo el funcionamiento de EOF en sockets, decidí crear un struct "especial" que le indicase al servidor que ya el cliente terminó de agregar notas al alta. ~~ESTE ESPECIAL~~ Entiendo que no es la manera más elegante de hacerlo, pero funciona. Asumo que las notas son positivas, por lo tanto ninguna nota en ningún DATA-T en arreglo-data podría tener como nota un número negativo. → FUNCIÓN igual que en pipes

```
DOCENTE
INT MAIN_CLIENTE () {
    // ...
    connect(sock);
    WRITE(sock, &consto, sizeof(consto)); // Envío el código de la materia, lo asumo conocido
    for (int i=0; i<k; i++) {
        WRITE(sock, &(arreglo-data[i]), sizeof(arreglo-data[i]));
    }
    // Envío un DATA-T especial para que el servidor termine el alta
    CHAR LU[8]; // No importa qué.
    DATA-T INVALIDO = { LU, -1 };
    WRITE(sock, &INVALIDO, sizeof(INVALIDO));
    CLOSE(sock);
}
```

```

INT MAIN-SERVIDOR () {
    // ...
    listen ( SERVIDOR, FD, N );
    while (1) {
        Client = accept ( SERVIDOR-FD );
        if (FORK() == 0) {
            CLOSE ( SERVIDOR-FD ); // No lleva numero en el control del hijo
            int codigo;
            ,0 < Toma un u-Parametro
            RECV ( CLIENT, &codigo, sizeof ( CODIGO ) );
            ACTA current = urutActa ( Codigo );
            DATA-T DATA-ACTA;
            while (1) {
                RECV ( CLIENT, &DATA-ACTA, sizeof ( DATA-ACTA ) );
                IF ( DATA-ACTA.NOTA == -1 ) { BREAK; } // El cliente termino de enviar
                AGREGARNota ( current, &( DATA-ACTA.LU ), DATA-ACTA.NOTA );
            }
            // Si salimos del while es porque el cliente pidió encender la acta
            // corriente Acta (current);
            close ( CLIENT );
            EXIT(0);
        } else {
            close ( Client );
        }
    }
}

```

25/25

Pedro Fuentes Urquiza
1088/22

44

Aula 2

2) // Bien Asumo que los siguientes variables están compartenas entre todos los procesos,
// incluyendo el central. También asumo que M y N son constantes.

SLOTS = SEM-INIT(1); ✓

MUTEX = SEM-INIT(1); ✓

PROCESADOS = 0; ✓

CONSOLIDAR = SEM-INIT(0); ✓

void Nodo_Central () {

while (1) {

CONSOLIDAR.WAIT();

CONSOLIDAR.DATASENSORES (); } ✓

}

void Proceso_Sensor () {

while (1) {

SLOTS.WAIT(); ✓

PREPAREDATASENSOR (); ✓

SLOTS.SIGNAL(); ✓

MUTEX.WAIT(); ✓

Sensor

PROCESADOS++;

if (PROCESADOS == N) {

PROCESADOS = 0;

CONSOLIDAR.SIGNAL();

}

MUTEX.SIGNAL(); ✓

}

{

3) a) Falso. Si es una única cola RR con el mismo quantum para todos los usuarios, no habría tal noción de prioridad, ya que todos serían un usuario de nivel 0 tendrían que esperar "lo mismo" para ejecutar que un usuario de nivel A-7.

3)b) Falso. Si suviéramos un algoritmo de prioridades ~~FIFO~~ sin feedback, los procesos de alta prioridad ejecutarseían todo el tiempo, dejando en ~~la~~ starvation a los de baja prioridad. Se puede proponer aging de los procesos con baja prioridad sin hacerlos volver.

Respecto del algoritmo dentro de cada cola, ~~se usa FIFO~~, ya que si un usuario se pone a ejecutar más durante mucho tiempo, ~~se pierde~~ dejar sin ejecutar a usuarios de su misma categoría (vía los de alto deadline). Por lo tanto, no tiene que ser FIFO.

3)c) Falso. Algunos sistemas de diseño lo harán muy breve. Como explicamos en el item anterior, se puede armar un diseño que otorgue mayor prioridad a cortos ciclos, pero que deje sin ejecutar a los largos.

② Se puede implementar ~~una~~ un feedback que sea las métricas de uso de CPU de los usuarios, que nos dan por categorías que tenemos. Si ese uso en el tiempo regular es muy bajo (cerca), se le da la categoría temporalmente, hasta que vuelve a ejecutar. Incluso, ese "threshold" de uso de CPU puede variar por categorías.

3). Verdadero. Si diseñara una multilevel que se describió en el ítem (1), ^{sin aging ni feedback} suceder que ~~desde~~ los usuarios de baja prioridad nunca acceden a la CPU, en una situación en la que los procesos de mayor prioridad (menor número de categoría) quieren ejecutar todo el tiempo. Como no sabemos la estructura del scheduler, no podemos garantizar que no haya starvation.

~~otra pregunta~~

3) b) Vemos que este enunciado es Falso, pues los características puntuales podrían hacer que no se garantice lo pedido en la concesión. En particular, si usamos FIFO para cada una de las colas de cada prioridad, se podrían dar una situación en la que un proceso de alta prioridad quiera ejecutar durante mucho tiempo,荒riendo muchos procesos como la cola es FIFO, no se le da el control a otro usuario hasta que el actual no termine. Esto ilustra la condición de "El sistema garantiza que los usuarios tienen la CPU de manera justa y entre que un usuario obtenga más programación implemente荒riendo más procesos".

Para solucionar esto, podríamos usar RR en cada una de las colas. De este modo, si un usuario荒riendo más procesos no se afecta su tiempo de ejecución, pues eventualmente se desaloja y otro usuario toma el control.

Para evitar荒riendo en los procesos de menor categoría, se puede implementar feed back, haciendo que un ~~usuario~~ ^{usuario} pueda salir de categoría. El criterio podría ser el ~~Tiempo~~ porcentaje de uso del CPU en el tiempo reciente, y el threshold para salir de categoría podría variar por categoría de usuario y por la categoría que está actualmente.

a) Vemos que el ~~array de páginas~~ array ocupa un total de 6 páginas.
La demanda de páginas accedida en la array es:

Σi Páginas accedidas

0	0, 1, 1, 0, 0
3	3, 8, 3, 3
6	6, 5, 6, 6
9	9, 2, 9, 9

También que:

$X(0)$	#1
$X(1)$	#2
$X(2)$	#3
$X(3)$	#4
$X(4)$	#5
$X(5)$	#6
$X(6)$	#7
$X(7)$	#8
$X(8)$	#9
$X(9)$	#10
$X(10)$	#11

luego, eso se traduce en las siguientes páginas.

$i=0$ $i=3$ $i=6$ $i=9$
 1, 6, 1, 1, 2, 3, 2, 2, 9, 3, 4, 4, 5, 2, 5, 5.

b) $F=3$ marcos de página. Averiguar qué páginas quedan varadas.

Página	LRU			Número de veces a la tasa de el tiempo de acceso)
	#1	#2	#3	
1 _x	1	-	-	1
6 _x	1	6	-	1, 6
1 _v	1	6	-	1, 6
1 _v	1	6	-	1, 6
2 _x	1	6	2	1, 6, 2
5 _x	1	9	2	1, 2, 1, 5
2 _v	1	5	2	2, 7, 5
2 _v	1	5	2	2, 7, 5
4 _x	4	5	2	5, 2, 4
3 _x	4	7	2	2, 4, 3
4 _v	4	7	2	3, 4, 3
4 _v	4	3	2	2, 4, 3
5 _x	4	3	5	4, 3, 5
2 _x	4	2	5	5, 4, 2
5 _v	4	2	5	5, 4, 2
5 _v	4	2	5	5, 4, 2
TOTAL	18	8	8	