Sistemas Operativos

Departamento de Computación – FCEyN – UBA

Primer cuatrimestre de 2022

Primer parcial – 12/5 - Primer cuatrimestre de 2022

Nombre y apellido: _____

Nº orden: _____ L.U.: _____ Cant. hojas: _____

1	2	3	4	Nota

- 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.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- **Cada código o pseudocódigo debe estar bien explicado y justificado en castellano. ¡Obligatorio!**
- 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 tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

Ejercicio 1.- Sincronización y concurrencia. **(20 puntos)**
Una fábrica de estantes requiere un número específico de materiales para fabricar cada estante. Cada uno requiere el siguiente tipo y cantidad específica de materiales (ni uno más, ni uno menos):

- Tablas: **N** tablas.
- Tornillos: **M** tornillos.
- Clavos: **K** clavos.

Para iniciar la fabricación de un estante, primero debe ensamblarse, y para esto deben estar todos los materiales requeridos. Para concluir la fabricación, también deben pintarse los estantes ensamblados, pero el proceso de pintura requiere que se pinten **L** estantes a la vez (ni uno más ni uno menos). El siguiente diagrama muestra el protocolo de fabricación: Recepción de materiales → Ensamblaje → Pintura → Despacho

Los materiales llegan a la fábrica constantemente, y el despacho de estantes también es constante.

Cree un programa (puede ser en pseudocódigo o C/C++) usando procesos concurrentes y herramientas de sincronización, que simule este comportamiento de fabricación, donde se tendrán 3 tipos de procesos que representarán a cada material. Además, no se podrá utilizar un proceso coordinador, sino que la coordinación debe darse entre los mismos procesos. Su programa debe asegurar que, cuando se inicia y se termina cada etapa, debe cumplir con el número exacto de materiales o estantes. Se deben evitar condiciones de carrera y *deadlocks* en su solución. *Hint*: Considere que 1 estante = **N** tablas + **M** tornillos + **K** clavos.

Ejercicio 2.- Scheduling. **(35 puntos)**
Se busca diseñar un *scheduler* para un sistema operativo que tiene que soportar, en principio, dos tipos de tareas: real time e interactivas. Para la primera iteración de este diseño, se debe proveer una cota para el peor caso del response time de una tarea real time; por el momento dichas tareas solo tendrán un único nivel de prioridad estático. Por el contrario, las tareas interactivas podrán pertenecer a una de k clases de prioridades. En este sentido, el sistema debería siempre favorecer a tareas con comportamiento intensivo en E/S. Sin embargo, también se espera que se pueda adaptar al cambio en el comportamiento de las tareas: si una tarea intensiva en CPU gradualmente comienza a volverse intensiva en E/S, habría que recompensarla. Análogamente, habría que penalizarlas si se diera el caso opuesto.

Por último, siempre es deseable intentar minimizar el costo de las operaciones de *scheduling*.

a) Proponer una estrategia de *scheduling* adecuada para el escenario propuesto. Es importante dejar en claro qué estructuras de datos del kernel van a necesitar, como se actualizarán y justificar cómo su algoritmo busca cumplir los objetivos de diseño.

b) ¿Podría darse una situación donde exista *starvation* para las tareas interactivas? ¿Y para las real time? Justificar.

Ejercicio 3.- Comunicación Inter-procesos **(30 puntos)**
Se cuenta con una operación computacional costosa que se desea repartir entre N subprocesos.

Para ello, el proceso padre dispone de una función *int dameNumero(int pid)* que dado el **PID** de un hijo le devolverá un número secreto. Este número secreto deberá ser enviado al hijo correspondiente utilizando pipes. Esta función solo puede ser llamada por el padre.

Cada subproceso deberá encargarse de realizar el cómputo del número correspondiente utilizando para ello la función *int calcular(int numero)*. El número que deben utilizar como parámetro es el resultado de la función *dameNumero* que el padre les envió.

Los subprocesos ejecutarán la función calcular y, a medida que vayan terminando, le informarán el resultado al padre.

El proceso padre deberá llamar a la función *void informarResultado(int numero, int resultado)*, la cual recibirá como parámetros el número sobre el que se ejecutó el cálculo, y el resultado que éste produjo. Esta función solamente podrá ser llamada desde el proceso padre.

La función *informarResultado* deberá ser llamada en el mismo orden en que los procesos fueron terminando los distintos cómputos.

Se implementó una versión de este programa, en la cual el proceso padre realiza polling sobre los hijos para ver si terminaron, es decir, los va recorriendo en orden y para cada hijo le pregunta si terminó, en caso de responder afirmativamente, llama a la función *informarResultado*.

```
void ejecutarHijo (int i, int pipes[][2]) {
    // ...
}
int main(int argc, char* argv[]){
    if (argc< 2) {
        printf ("Debe ejecutar con la cantidad de hijos como parametro\n");
    return 0; }
    int N = atoi(argv[1]);
    int pipes[N*2][2];
    for ( int i=0; i< N*2; i++){
        pipe(pipes[i]); }
    for (int i=0; i< N; i++) {
        int pid = fork () ;
        if (pid==0) {
            ejecutarHijo(i,pipes);
            return 0;
        } else {
            int numero = dameNumero(pid) ;
            write(pipes[i][1], &numero, sizeof(numero)); } }
    int cantidadTerminados = 0;
    char hijoTermino [N] = {0};
    while (cantidadTerminados < N) {
        for ( int i=0; i< N; i++) {
            if (hijoTermino[i]) {
            continue; }
            char termino = 0;
            write(pipes[i][1], &termino, sizeof(termino));
            read(pipes[N+i][0], &termino, sizeof(termino));
            if (termino) {
                int numero;
                int resultado ;
                read(pipes[N+i][0], &numero, sizeof(numero));
                read(pipes[N+i][0], &resultado, sizeof(resultado));
                informarResultado(numero, resultado);
                hijoTermino[i] = 1;
                cantidadTerminados++; }  }  }
    wait(NULL) ;
    return 0; }
```

Resolver la función *ejecutarHijo()* utilizando pipes y señales, respetando el siguiente comportamiento. Para poder responder al *polling* del padre, cada hijo deberá crear un segundo subproceso que será el encargado de ejecutar la función *calcular*. Este subproceso (nieto) le avisará a su padre cuando haya terminado mediante una señal, comunicándole además el resultado. El proceso hijo una vez que sepa que su proceso nieto terminó, responderá afirmativamente al polling del padre, enviándole el número y el resultado. A efectos del ejercicio y para evitar las posibles condiciones de carrera ocasionadas por el polling, se asumirá que dos llamados concurrentes a la función calcular no pueden terminar a la vez ni tampoco cercanos en el tiempo, sino con varios minutos de diferencia entre uno y otro.

Ejercicio 4.- Gestión de memoria: **(15 puntos)**
Considere la siguiente secuencia de referencias a páginas: $1, 2, 8, 1, 4, 2, 7, 9, 4, 5$

a) Realice el seguimiento de cada uno de los algoritmos de reemplazo listados abajo, considerando que el sistema cuenta con 4 *frames* (todos ellos inicialmente libres). Además, indique el *hit-rate* para cada algoritmo.

   i. FIFO.
   ii. LRU.
   iii. Segunda oportunidad.

b) Para cada algoritmo de reemplazo de páginas del ítem anterior con menor *hit-rate*, exhiba un escenario (secuencia de páginas) donde su *performance* (*hit-rate*) sea superior a los otros dos.