

Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de Ciencias Computacionales

Sistemas Operativos



Profesora: Becerra Velázquez Violeta del Rocío

Alumno: Monjaraz Briseño Luis Fernando

Código: 218520958

Carrera: Ingeniería en Computación

Sección: D04

Actividad 5

Fecha: 24/09/2023

Índice

Índice	2
Tabla de imágenes	3
Datos personales	4
Datos de la materia	4
Número de actividad	4
Objetivo de la actividad	4
Notas acerca del lenguaje	4
Conclusión	14

Tabla de imágenes

<i>Ilustración 1 Struct</i>	5
<i>Ilustración 2 Uso de rand</i>	6
<i>Ilustración 3 Kbhit</i>	7
<i>Ilustración 4 Código</i>	8
<i>Ilustración 5 Código 2</i>	9
<i>Ilustración 6 ValidaNumerosEnteros</i>	9
<i>Ilustración 7 Gotoxy</i>	10
<i>Ilustración 8 IWillHaveOrder</i>	10
<i>Ilustración 9 ThisIsOrder</i>	11
<i>Ilustración 10 datosLotes</i>	12
<i>Ilustración 11 imprimirdata</i>	13
<i>Ilustración 12 main</i>	13
<i>Ilustración 13 Ejemplo de cómo debería verse</i>	14
<i>Ilustración 14 Segundo Ejemplo</i>	15

Datos personales

Nombre: Monjaraz Briseño Luis Fernando

Código: 218520958

Correo: luis.monjaraz5209@alumnos.udg.mx

Datos de la materia

Materia: Sistemas Operativos

Sección: D04

Horario: Martes, Jueves, Sábado. 11:00 a 12:55

NRC: 204880

Clave: IL366

Número de actividad

Programa 2. Simular el procesamiento por lotes con Multiprogramación.

Objetivo de la actividad

El objetivo de la actividad es la recrear el funcionamiento de un procesamiento por lotes, el cual consta de ejecutar un lote el cual contiene una cantidad de procesos, dichos procesos solo se podrán realizar en orden, o sea, el primero termina primero y el ultimo termina último, al limitar la ejecución de los procesos a un orden específico, se refleja cómo funcionaban los sistemas en esa época y cómo la automatización de tareas en lotes permitía mantener un flujo constante de trabajo sin intervención manual constante. En esta actividad se realizan 5 procesos por lote, a este le asignamos una cantidad de tiempo estimado, esto nos ayuda a comprender el proceso en el que el sistema asigna los recursos para el proceso, al dividirlo en lotes también observamos el como se maneja la transición de cada lote (como se dice en la teoría del procesamiento por lotes). Además de entender y practicar el uso de “kbhit” para poder realizar pausas, interrupciones, errores y continuar, esto para que sea todavía mas inmersivo y recree de mejor manera estos procesos por lotes.

En resumen, esta actividad es sumamente útil para comprender la teoría del procesamiento por lotes al igual que darnos un vistazo al pasado de las computadoras y ponernos en los pies de aquellos que trabajaron en dichas computadoras, al igual que entender el funcionamiento interno de un sistema y el uso de kbhit.

Notas acerca del lenguaje

Lenguaje usado: C++

Motivo: Al ser una “actualización” del programa anterior la mejor opción era volver a usar C++, esto para realizar las modificaciones apropiadas para que cumpla con lo solicitado, esto también tiene que ver a que utilice estructuras de datos las cuales solamente e practicado en C++, al igual de que es el lenguaje que mas usamos en la carrera así que me sirve para seguir practicándolo. Pero en general el motivo principal es el primero, o sea, reutilizar el anterior

código. Aunque cabe resaltar que para las impresiones utilice gotoxy y creo yo que es lo que mas complico el código, pues estar cuidando que cada cosa se este imprimiendo y actualizando constantemente fue una experiencia por así decirlo “curiosa”.

Estructuras: Utilice un “Struct” llamado “Process” en esta se almacenan los datos que se utilizaran para los procesos, la estructura es una cola estática (limitada a 5), para que siempre existan mas “Lotes” que procesos hice que la cantidad de colas sea equivalente a la cantidad de procesos, por lo que siempre se podrán almacenar todos los procesos.

```

34  struct Process {
35      string operation;
36      int number1;
37      int number2;
38      int result;
39      int estimatedTime;
40      int programNumber;
41      int currentQueue;
42      int tiempoTranscurrido; // no se aplica en todos los casos
43      int tiempoRestante; // no se aplica en todos los casos
44  };
45

```

Ilustración 1 Struct

Funciones:

- ValidaNumerosEnteros: Esta función valida que lo que el usuario ingrese sea un numero entero, esto mediante un char para facilitar su comparación.
- Gotoxy: Esta me ayuda a hacer el aspecto visual del programa.
- IWillHaveOrder: Es la función que grafica las líneas visuales de los datos.
- ThisIsOrder: Es la función que grafica las líneas visuales del programa.
- DatosLotes: Es la función encargada de solicitar y hacer los cálculos con los datos de los procesos, al igual que validarlos.
- ImprimirDatos: Es la función encargada de imprimir toda la información, por lo que también es la que imprime el tiempo.
- Main: Es la función principal que llama a las demás y realiza limpieza en la pantalla.

Al ser una “actualización” del programa anterior la estructura es prácticamente la misma, o sea, se utilizó una cola estática para simular el funcionamiento de los lotes las cuales están limitadas a 5 “procesos”, donde el usuario ingresa la cantidad de procesos y el sistema elegirá todos los demás valores de forma aleatoria, en este caso use “Rand()”, este no es completamente aleatorio pues este se considera “pseudoaleatorio” pero a final de cuentas ni yo ni el usuario determina los valores. Ejemplo del uso de “Rand()”:

```
for (int i = 1; i <= totalProcesses; ++i) {
    Process newProcess;
    system("cls");
    ThisIsOrder();
    newProcess.currentQueue = currentQueue+1;
    int operationIndex = rand() % 6;
    switch (operationIndex)
    {
        case 0:
            newProcess.operation = "+";
            break;
        case 1:
            newProcess.operation = "-";
            break;
        case 2:
            newProcess.operation = "*";
            break;
        case 3:
            newProcess.operation = "/";
            break;
        case 4:
            newProcess.operation = "residuo";
            break;
        case 5:
            newProcess.operation = "porcentaje";
            break;
        default:
            break;
    }
    int operationN1 = rand() % 101;
    int operationN2 = rand() % 101;
    newProcess.number1 = operationN1;
    newProcess.number2 = operationN2;

    if(newProcess.operation == "/"){
        while(newProcess.number2 == 0){
            int operationN2 = rand() % 101;
```

Ilustración 2 Uso de rand

Para dar la “ilusión” de que se tarda utilice Sleep y gotoxy para que se muestre una vez que terminara el tiempo de ejecución, en general, eso aplico para toda la impresión, pues con la ayuda de Sleep controle los tiempo para que una vez que pasara el tiempo máximo estimado este mostrara los resultados, al igual que en estos tiempos se pueden realizar las interrupciones, pausas y errores, y con la ayuda de gotoxy le di forma a todo el código, es verdad que esto trae problemas pero nada que no se solucione calculando correctamente las coordenadas al igual que

borrar correctamente las mismas. Realmente no tiene mucha ciencia el código, o sea, si tengo muchas declaraciones pero estas son para controlar en su mayoría el tiempo o la cantidad de procesos y lotes, pues para calcular la cantidad de lotes tuve que realizar una operación matemática, esto debido a que técnicamente la cantidad de lotes que puedo ingresar la limite a la cantidad de procesos que ingrese el usuario, si lo sé, puede ser confuso pero básicamente así me aseguro de que siempre existan mas lotes, pues si el usuario ingresa 5 procesos estos solo ocuparían un lote pero realmente se generaron 5, aunque los otros 4 estén en desuso, lo mismo pasaría con 10 procesos, se generarían 10 lotes pero solo 2 estarían en uso, algo rebuscado pero en su momento fue lo que mejor se me ocurrió.

Importante: En cuanto habrá el programa (importante que sea desde el .exe)dele al botón de “Maximizar pantalla”, o sea, el cuadradito que está en medio de minimizar y cerrar, esto porque se emplea Gotoxy y este ocasiona problemas si la pantalla no es lo suficientemente grande, de igual forma el Gotoxy está adaptado a mi pantalla (14 pulgadas) por lo que, si su pantalla es menor no se verá bien, si esta es mayor si se verá bien. De todas formas, implemente unas líneas de código que vuelven la pestaña más grande de lo normal. Nota: La cantidad de librerías es porque son de “colección” de librerías, por lo que no se usaron todas.

Cambios principales para la interrupción, pausa, errores y continuar:

```

370 if (_kbhit() && process.tiempotranscurrido != process.estimatedTime) {
371     char key = _getch();
372     if (key == 'I' || key == 'i') { // interrumpir
373         pulsar = 'i';
374         interruptedProcess.tiempotranscurrido = process.tiempotranscurrido; // Guardar el tiempo transcurrido
375         interruptedProcess.tiemporestante = process.tiemporestante; // Guardar el tiempo total
376         queues[i].pop(); // Sacar el proceso de la cola actual
377         queues[i].push(interruptedProcess);
378         gotoxy(18,15);
379         cout << " ";
380         timecontador = timecontador - 1;
381         queue<Process> tempQueue = queues[i]; // Copia temporal de la cola
382         y = 7;
383         ay = 7;
384         while (!tempQueue.empty()) {
385             // fila 1
386             Process tprocess = tempQueue.front();
387             tempQueue.pop();
388             gotoxy(3,5);
389             cout << "ID";
390             gotoxy(3,y);
391             cout << " ";
392             gotoxy(3,y);
393             cout << tprocess.programNumber;
394             gotoxy(6,5);
395             cout << "TIME";
396             gotoxy(6,y);
397             cout << " ";
398             gotoxy(6,y);
399             cout << tprocess.estimatedTime;
400             y = y + 2;
401         }
402         break;
403     } else if (key == 'P' || key == 'p') { // pausar
404         while (true)
405             gotoxy(80,21);
406             cout << "Pausado";
407             {
408                 if (_kbhit()) {
409                     char key = _getch();
410                     if (key == 'C' || key == 'c') { // continuar
411                         gotoxy(80,21);
412                         cout << " ";
413                         break;
414                     }
415                 }
416             }
417     } else if (key == 'E' || key == 'e') { // error
418         pulsar = 'e';
419         timecontador = timecontador - 1;
420         break;
421     }
422 }
423 Sleep(1000);

```

Ilustración 3 Kbhit

Simular el procesamiento por lotes con Multiprogramación

Como se puede apreciar la mayoría de los if cambian un valor de “pulsar” este es para que al momento de la impresión se seleccione la forma correcta dependiendo de la opción.

```
436         if(pulsar == 'e'){
437             gotoxy(34,ffiy);
438             cout << " ";
439             gotoxy(34,ffiy);
440             cout << process.programNumber;
441             gotoxy(38,ffiy);
442             cout << " ";
443             gotoxy(38,ffiy);
444             cout << process.number1 << process.operation << process.number2;
445             gotoxy(60,ffiy);
446             cout << " ";
447             gotoxy(60,ffiy);
448             cout << "ERROR";
449             gotoxy(66,ffiy);
450             cout << " ";
451             gotoxy(74,ffiy);
452             cout << process.currentQueue;
453             ffiy = ffiy + 2;
454             pulsar = ' ';
455             // acomodar fila 1
456             gotoxy(3,ay);
457             cout << " ";
458             ay = ay + 2;
459             queues[i].pop();
460             procesosTotales = procesosTotales - 1;
461         }
462         else if (pulsar == 'i')
463         {
464             if(process.estimatedTime == process.tiempotranscurrido){
465                 gotoxy(34,ffiy);
466                 cout << " ";
467                 gotoxy(34,ffiy);
468                 cout << process.programNumber;
469                 gotoxy(38,ffiy);
470                 cout << " ";
471                 gotoxy(38,ffiy);
472                 cout << process.number1 << process.operation << process.number2;
473                 gotoxy(60,ffiy);
474                 cout << " ";
475                 gotoxy(60,ffiy);
476                 cout << process.result;
477                 gotoxy(66,ffiy);
478                 cout << " ";
479                 gotoxy(74,ffiy);
480                 cout << process.currentQueue;
481                 ffiy = ffiy + 2;
482                 // acomodar fila 1
483                 gotoxy(3,ay);
484                 cout << " ";
```

Ilustración 4 Código


```

470         cout << " ";
471         gotoxy(38,ffty);
472         cout << process.number1 << process.operation << process.number2;
473         gotoxy(60,ffty);
474         cout << " ";
475         gotoxy(60,ffty);
476         cout << process.result;
477         gotoxy(66,ffty);
478         cout << " ";
479         gotoxy(74,ffty);
480         cout << process.currentQueue;
481         ffty = ffty + 2;
482         // acomodar fila 1
483         gotoxy(3,ay);
484         cout << " ";
485         ay = ay + 2;
486         queues[i].pop();
487         procesosTotales = procesosTotales - 1;
488     }
489     else{
490     }
491 }
492 else {
493     gotoxy(34,ffty);
494     cout << " ";
495     gotoxy(34,ffty);
496     cout << process.programNumber;
497     gotoxy(38,ffty);
498     cout << " ";
499     gotoxy(38,ffty);
500     cout << process.number1 << process.operation << process.number2;
501     gotoxy(60,ffty);
502     cout << " ";
503     gotoxy(60,ffty);
504     cout << process.result;
505     gotoxy(66,ffty);
506     cout << " ";
507     gotoxy(74,ffty);
508     cout << process.currentQueue;
509     ffty = ffty + 2;
510     // acomodar fila 1
511     gotoxy(3,ay);
512     cout << " ";
513     ay = ay + 2;
514     queues[i].pop();
515     procesosTotales = procesosTotales - 1;
516 }
517 }

```

Ilustración 5 Código 2

Funciones:

```

47 bool ValidaNumerosEnteros(char *dato){
48     bool ban = true;
49     int i = 0;
50     if (*dato == '-' || *dato == '+') {
51         i++;
52     }
53     while (*(dato + i) != '\0') {
54         if (*(dato + i) < '0' || *(dato + i) > '9') {
55             ban = false;
56             break;
57         }
58         i++;
59     }
60     return ban;
61 }

```

Ilustración 6 ValidaNumerosEnteros

```
63 void gotoxy(int x,int y){
64     HANDLE hcon;
65     hcon = GetStdHandle(STD_OUTPUT_HANDLE);
66     COORD dwPos;
67     dwPos.X = x;
68     dwPos.Y= y;
69     SetConsoleCursorPosition(hcon,dwPos);
70 }
71
```

Ilustración 7 Gotoxy

```
void IWillHaveOrder(){
    int x = 1, y = 1;
    gotoxy(0,0);
    printf("%c", 201); //┐
    gotoxy(132,0);
    printf("%c", 187); //┘
    gotoxy(0,31);
    printf("%c", 200); //└
    gotoxy(132,31);
    printf("%c", 188); //┘
    while (y<=30)
    {
        gotoxy(0,y);
        printf("%c", 186); //||
        gotoxy(132,y);
        printf("%c", 186); //||
        y++;
    }
    while (x<=131)
    {
        gotoxy(x,0);
        printf("%c", 205); //=-
        gotoxy(x,31);
        printf("%c", 205); //=-
        x++;
    }
    y = 1;
    int y2 = 3;
    gotoxy(11,2);
    printf("%c", 203); //┌
    gotoxy(11,31);
    printf("%c", 202); //└
}
```

Ilustración 8 IWillHaveOrder

```
46  void ThisIsOrder(){
47      int x = 1, y = 1;
48      gotoxy(0,0);
49      printf("%c", 201); // 
50      gotoxy(132,0);
51      printf("%c", 187); // 
52      gotoxy(0,31);
53      printf("%c", 200); // 
54      gotoxy(132,31);
55      printf("%c", 188); // 
56      while (y<=30)
57      {
58          gotoxy(0,y);
59          printf("%c", 186); // 
60          gotoxy(132,y);
61          printf("%c", 186); // 
62          y++;
63      }
64      while (x<=131)
65      {
66          gotoxy(x,0);
67          printf("%c", 205); // 
68          gotoxy(x,31);
69          printf("%c", 205); // 
70          x++;
71      }
72  }
73
```

Ilustración 9 ThisIsOrder

```
174 void datosLotes(){
175     char totalProcessesc[100];
176     gotoxy(1,1);
177     cout << "Ingrese el numero de procesos: ";
178     cin >> totalProcessesc;
179     while(!ValidaNumerosEnteros(totalProcessesc)){
180         gotoxy(1,1);
181         cout << "
182         gotoxy(1,1);
183         cout << "Ingrese el numero de procesos de nuevo: ";
184         cin >> totalProcessesc;
185     }
186     totalProcesses = atoi(totalProcessesc);
187     totallotes = totalProcesses % 5;
188     if (totallotes == 0) {
189         totallotes = 0;
190     } else {
191         totallotes = totalProcesses + (5 - totallotes);
192     }
193     totallotesr = totallotes / 5;
194     totallotesrr = totallotes - (totallotesr*4);
195
196     int currentQueue = 0;
197     for (int i = 1; i <= totalProcesses; ++i) {
198         Process newProcess;
199         system("cls");
200         ThisIsOrder();
201         newProcess.currentQueue = currentQueue+1;
202         int operationIndex = rand() % 6;
203         switch (operationIndex)
204         {
205             case 0:
206                 newProcess.operation = "+";
207                 break;
208             case 1:
209                 newProcess.operation = "-";
210                 break;
211             case 2:
```

Ilustración 10 datosLotes

```

280
281 void imprimirdata() {
282     int procesosTotales = totalProcesses;
283     int lotesf = totalLotesrr;
284     int ffy = 7;
285     char pulsar = ' ';
286     gotoxy(80, 13);
287     cout << "I = Interrumpir";
288     gotoxy(80, 15);
289     cout << "E = Error";
290     gotoxy(80, 17);
291     cout << "P = Pausar";
292     gotoxy(80, 19);
293     cout << "C = Continuar";
294     for (int i = 0; i < maxQueues; i++) {
295         gotoxy(1,1);
296         cout << "Lotes restantes: " << lotesf;
297         int ay = 7;
298         int y = 7;
299         if (!queues[i].empty()) {
300             gotoxy(3,3);
301             cout << "Lote #" << i + 1 << endl;
302             queue<Process> tempQueue = queues[i]; // Copia temporal de la cola
303             while (!tempQueue.empty()) {
304                 // fila 1
305                 Process tprocess = tempQueue.front();
306                 tempQueue.pop();
307                 gotoxy(3,5);
308                 cout << "ID";
309                 gotoxy(3,y);
310                 cout << tprocess.programNumber;
311                 gotoxy(6,5);
312                 cout << "TME";
313                 gotoxy(6,y);
314                 cout << tprocess.estimatedTime;
315                 y = y + 2;
316             }
317             while (!queues[i].empty()){

```

Ilustración 11 imprimirdata

```

543
544 int main() {
545     HWND consoleWindow = GetConsoleWindow();
546     RECT desktop;
547     GetWindowRect(GetDesktopWindow(), &desktop);
548     MoveWindow(consoleWindow, desktop.left, desktop.top, desktop.right, desktop.bottom, TRUE);
549     system("pause");
550     ThisIsOrder();
551     datosLotes();
552     system("cls");
553     IWillHaveOrder();
554     imprimirdata();
555     gotoxy(80,30);
556     system("pause");
557     system("cls");
558     return 0;
559 }
560

```

Ilustración 12 main

Conclusión

En conclusión, esta actividad es una clara forma de representar el funcionamiento de sistemas antiguos, en el caso de la programación, debo decir que esta fue mucho mas sencilla que la anterior, esto debido a que la mayoría de las cosas fueron modificaciones y no crear el código desde cero. El uso de kbhit fue interesante para la aplicación de este trabajo, ya que nos permite junto un getch capturar las teclas que una persona ingreso para determinar si quiere interrumpir, pausar, continuar o darle error al programa, esto sin detener directamente el while que utilice, pero pues en lo personal considero que hay mejores opciones sin la necesidad de implementar un kbhit, aunque para esto si se necesitaría una interfaz gráfica.

Nota: Se realizaron correcciones en base a los comentarios de la clase.

Como se debería de ver:

Lote #1	Ejec
ID TME TT	Nn
3 6 2	ID 2
4 16 1	Ope porcentaje
5 8 2	TME 10
1 12 10	TT 8
	TR 2

Terminados

ID Ope Res Lote Procesos restantes
5
Tiempo total
17
I = Interrumpir
E = Error
P = Pausar
C = Continuar

Ilustración 13 Ejemplo de cómo debería verse

D:\beto5\Descargas\Tareas\SO\Act5 SO\MonjarazLuisD04Act5\output\MonjarazLuisD04Act5.exe

Lotes restantes: 0								
Lote #1	Ejec		Terminados					
ID TME TT	Nn		ID Ope	Res	Lote	Procesos restantes		
	ID 1		2 69porcentaje65	44	1			
			3 22residuo49	22	1			
	Ope porcentaje		4 61-63	-2	1	Tiempo total		
			5 24-80	-56	1			
	TME 12		1 85porcentaje72	61	1	45		
	T0 11					I = Interrumpir		
	T0 1					E = Error		
						P = Pausar		
						C = Continuar		
						Presione una tecla para continuar . . .		

Ilustración 14 Segundo Ejemplo

Enlace de descarga:

<https://drive.google.com/drive/folders/1MKvyamRmcJovvQuiZHfiimgr95LOp-TL?usp=sharing>