



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA CUATRO

Simulación del algoritmo de Round Robin

Alumno:

Rojas Zepeda Luis Eduardo

Profesor:

Velez Saldaña Ulises

2CM6, 29/03/19



Índice

1. Teoría	3
2. Material	6
3. Desarrollo	7
4. Bibliografía	17

1. Teoría

Algoritmo de Round Robin

Round-robin es un método para seleccionar todos los abstractos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento. El nombre del algoritmo viene del principio de Round-Robin conocido de otros campos, donde cada persona toma una parte de un algo compartido en cantidades, es decir, "toma turnos". En operaciones computacionales, un método para ejecutar diferentes procesos de manera concurrente, para la utilización equitativa de los recursos del equipo, es limitando cada proceso a un pequeño período (quantum), y luego suspendiendo este proceso para dar oportunidad a otro proceso y así sucesivamente. A esto se le denomina comúnmente como Planificación Round-Robin.

Los sistemas implementados con ese algoritmo de planificación son llamados de tiempo compartido ya que el tiempo de procesador se ve como un recurso que se comparte por turnos entre los procesos. El funcionamiento del algoritmo es dar un rodaja de CPU a cada proceso de forma secuencial, la selección entre procesos activos se gestiona según el que lleve mas tiempo esperado(FIFO).

Características

1. Cada proceso tiene un quantum(q)
2. La cola de procesos es circular y se gestiona como FIFO.
3. N es el numero de procesos
4. El tiempo de respuesta máximo es $q(N-1)$

Conceptos

1. Quantum: Es el numero máximo de intervalos de tiempo que un proceso puede utilizar la CPU.
2. Tiempo de llegada(TL): Es el intervalo de tiempo en que comienza el proceso.
3. Tiempo de ejecución o Rafaga(TE): Es el intervalo de tiempo que demora el proceso en ejecutarse.
4. Tiempo de finalizacion(TF): Es el intervalo de tiempo en que termino el proceso.
5. Tiempo de retorno(TR): Es la suma de intervalos de tiempo que pasa desde que el proceso se lanza hasta que finaliza su ejecución. Se prodria de es la suma del tiempo de espera mas el tiempo de ejecución.
6. Tiempo de espera(TES): Es la suma de intervalos de tiempo que esta un proceso en la cola de procesos listo.
7. Tiempo de respuesta(TRS): Tiempo que pasa desde que se manda a ejecuta el proceso hasta que se ejecuta por primera vez.

TDA Un Tipo de dato abstracto (en adelante TDA) es un conjunto de datos u objetos al cual se le asocian operaciones. El TDA provee de una interfaz con la cual es posible realizar las operaciones permitidas, abstrayéndose de la manera en como estén implementadas dichas operaciones. Esto quiere decir que un mismo TDA puede ser implementado utilizando distintas estructuras de datos y proveer la misma funcionalidad.

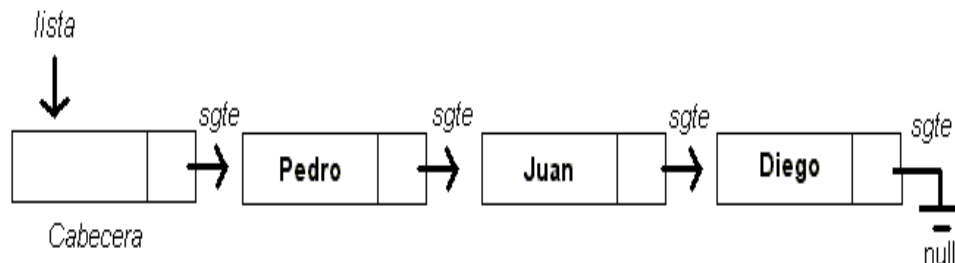
Listas Una lista se define como una serie de N elementos E_1, E_2, \dots, E_N , ordenados de manera consecutiva, es decir, el elemento E_k (que se denomina elemento k -ésimo) es previo al elemento E_{k+1} . Si la lista contiene 0 elementos se denomina como lista vacía.

Las operaciones que se pueden realizar en la lista son: insertar un elemento en la posición k , borrar el k -ésimo elemento, buscar un elemento dentro de la lista y preguntar si la lista esta vacía.

Las operaciones básicas son:

1. `estaVacía()`: devuelve verdadero si la lista esta vacía, falso en caso contrario.
2. `insertar(x, k)`: inserta el elemento x en la k -ésima posición de la lista.
3. `buscar(x)`: devuelve la posición en la lista del elemento x .
4. `buscarK(k)`: devuelve el k -ésimo elemento de la lista.
5. `eliminar(x)`: elimina de la lista el elemento x .

La lista se puede representar de la siguiente manera



Cola circular Una cola circular o anillo es una estructura de datos en la que los elementos están de forma circular y cada elemento tiene un sucesor y un predecesor. Los elementos pueden consultarse, añadirse y eliminarse únicamente desde la cabeza del anillo que es una posición distinguida. Existen dos operaciones de rotaciones, una en cada sentido, de manera que la cabeza del anillo pasa a ser el elemento sucesor, o el predecesor, respectivamente, de la cabeza actual.

Makefile Make es una herramienta de gestión de dependencias, típicamente, las que existen entre los archivos que componen el código fuente de un programa, para dirigir su recompilación o "generación" automáticamente. Si bien es cierto que su función básica consiste en determinar automáticamente qué partes de un programa requieren ser recompiladas y ejecutar los comandos necesarios para hacerlo, también lo es que Make puede usarse en cualquier escenario en el que se

requiera, de alguna forma, actualizar automáticamente un conjunto de archivos a partir de otro, cada vez que éste cambie

La estructura básica es la siguiente:

<pre>objetivo: dependencias comandos</pre>
--

En “objetivo” definimos el módulo o programa que queremos crear, después de los dos puntos y en la misma línea podemos definir qué otros módulos o programas son necesarios para conseguir el “objetivo”. Por último, en la línea siguiente y sucesivas indicamos los comandos necesarios para llevar esto a cabo. Es muy importante que los comandos estén separados por un tabulador del comienzo de línea.

2. Material

Editor de Texto Nano

Solo se instala con la siguiente linea:

```
sudo apt-get install nano
```

Copilador gcc

Se instala con la siguiente linea:

```
sudo apt-get install gcc
```

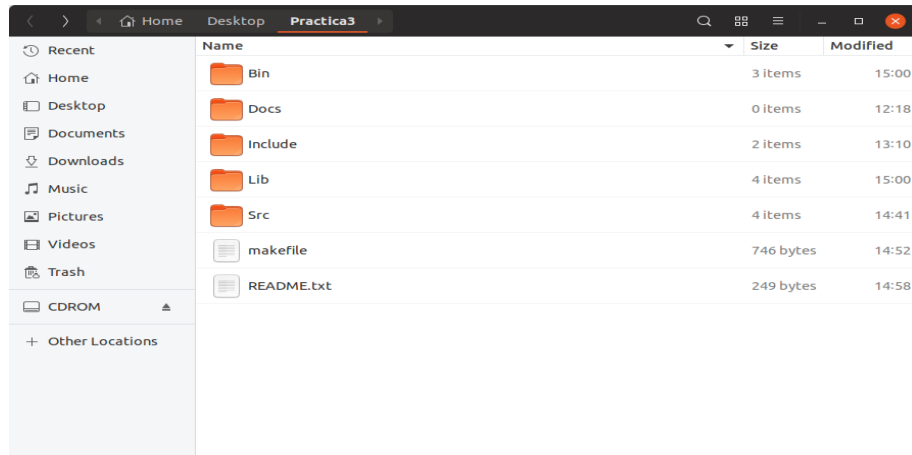
Makefile

Se instala con la siguiente linea:

```
sudo make install
```

3. Desarrollo

Crearemos las siguientes carpetas:



Después se debe generar le siguiente códigos que es el algoritmo de Round Robin:

```

/*
\mainpage
\author: Rojas Zepeda Luis Eduardo
\version 1.0
\date March 31 2019

*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "../Include/Cola.h"
#include "../Include/Lista.h"

/**Cabecera funcion listaJobs*/
Lista listaJobs();
/**Cabecera funcion imprimeCola*/
void imprimeCola(Cola);

/**\brief Programa principal. */
int main(int argc, char *argv[]){

/**

\details
Una vez que se obtienen la lista de procesos y la cola de preprocesos
, se simula el algoritmo de Round Robin

*/

int chunk=0;
int tiempo=0;

```

```

int contador=0;
Lista jobs = listaJobs();
Cola ready = nueva();
//impLista(jobs);

printf("Chunk:_");
scanf("%d", &chunk);

/** \ details
Algoritmo de Round Robin
*/
while (!esVacia(jobs) || !esNueva(ready)){
    if (!esVacia(jobs))
        if (horaLlegada(jobs) == tiempo){
            ready = formar(ready, idProceso(jobs),
                            horaLlegada(jobs), duracion(jobs));
            printf("\n%d_Aqui_estoy_(se_formo_este_
                    preoceso)\n\n", idProceso(jobs));
            imprimeCola(ready);
            jobs = resto(jobs);
        }
        if (esNueva(ready))
            printf("No_ha_llegado_ningun_proceso\n");
    else{
        if (contador<chunk){
            if (duracionC(ready)!=0){
                printf("Soy_el_proceso_%d\n",
                        idProcesoC(ready));
                restaDuracion(ready);
                //printf(" duracion: %d\n\n",
                        duracionC(ready));
            } else{
                printf("\n");
                printf("%d_Adios\n\n", idProcesoC(
                    ready));
                if (!esNueva(ready))
                    ready = desformar(ready);
            }
        } else{
            contador=-1;
            //printf("rota proceso\n");
            if (!esNueva(ready))
                ready = rotar(ready);
        }
        contador++;
        tiempo++;
    }
}

return 0;
}

/*      *\ brief Imprime en formato de tabla la cola de procesos
        actual.
*\param ready Es una cola con los procesos listos para ejecutare
        esperando su turno.
*/
void imprimeCola(Cola ready){
    printf("IdProceso\tHoraLlegada\tDuracion\n");
    impCola(ready);
    printf("\n");
}

```



```

}

/*      *\brief Obtiene de un archivo de texto los procesos con sus
        respectivos valores y los enlista segun su tiempo de llegada
*\return jobs Es una lista con todos los procesos y sus atributos
        ordenados segun su hora de llegada
*/
Lista listaJobs() {

    char pal[80], *w;
    char delim[] = ", ";
    FILE *fp;
    int idProceso = 0, horaLlegada = 0, duracion = 0;
    Lista jobs = vacia();
    int i;

    fp = fopen ( "procesos.txt", "r" );
    if (fp==NULL) {
        fputs ("File_error", stderr);
        exit (1);
    }

    while (fscanf(fp, "%s", pal) != EOF) {
        w=(char*)malloc(sizeof(char)*10);
        strcpy(w, pal);
        char *ptr = strtok(pal, delim);
        i=0;
        while(ptr != NULL)
        {
            if(i==0) idProceso = atoi(ptr);
            if(i==1) horaLlegada = atoi(ptr);
            if(i==2) duracion = atoi(ptr);
            ptr = strtok(NULL, delim);
            i++;
        }
        jobs = InsOrd(idProceso, horaLlegada, duracion, jobs);
    }
    fclose ( fp );

    return jobs;
}

```

Para el funcionamiento del código se requieren los TDA lista y cola circular. Dichos códigos estan acontinuacion.

TDA lista:

```

/*
\libreria del TDA lista
\author: Rojas Zepeda Luis Eduardo
\version 1.0
\date March 31 2019

*/

typedef int Elem;

/**
\brief Estructura que funciona como un nodo de la lista
        enlazada en un solo sentido, contiene los atributos de
        cada proceso
y un apuntador al nodo del siguiente proceso.
*/

```

```
typedef struct Nodo{
    Elem idProceso;
    Elem horaLlegada;
    Elem duracion;
    struct Nodo *sig;
}*Nodo;

typedef Nodo Lista;

/**
\brief Inicializa una lista vacia.
\return NULL representa el valor vacio de la lista.
*/
Lista vacia(){
    return NULL;
}

/**
\brief Funcion observadora que permite saber si la Lista
    esta vacia.
\param l Lista que se desea observar.
\return int entero con valor 0 para falso y algo diferente
    para verdadero.
*/
int esVacia(Lista l){
    return l==NULL;
}

/**
\brief Funcion que permite agregar elementos a la
    estructura.
\param idProceso Identificador del proceso.
\param horaLlegada Hora de llegada del proceso.
\param duracion Duracion del procesos.
\param l Lista a la que se formara el proceso.
\return t Lista con el proceso agragado.
*/
Lista cons(Elem idProceso, Elem horaLlegada, Elem duracion,
    Lista l){
    Lista t=(Lista)malloc(sizeof(struct Nodo));
    t->idProceso = idProceso;
    t->horaLlegada = horaLlegada;
    t->duracion = duracion;
    t->sig=l;
    return t;
}

/**
\brief Funcion que permite acceder al identificador del
    primer elemento de la lista.
\param l Lista que se consultara.
\return idProceso Identificador del proceso consultado.
*/
Elem idProceso(Lista l){
    return l->idProceso;
}

/**
\brief Funcion que permite acceder a la hora de llegada del
    primer elemento de la lista.
\param l Lista que se consultara.
\return horaLlegada Hora de llegada del proceso consultado.
```

```
*/
Elem horaLlegada(Lista l){
return l->horaLlegada;
}

/**
\brief Funcion que permite acceder a la duracion del primer
        elemento de la lista.
\param l Lista que se consultara.
return duracion Duracion del proceso consultado.
*/
Elem duracion(Lista l){
return l->duracion;
}

/**
\brief FUnction que desenlista al primer elemento de la
        lista.
\param l Lista que modificara.
return l->sig lista con el primer elemento retirado.
*/
Lista resto(Lista l){
return l->sig;
}

/**
\brief Imprime un dato.
\param e Dato que se desea imprimir.
*/
void impElem(Elem e){
printf("%d\n",e);
}

/**
\brief Imprime una lista.
\param l Lista que desea imprimir.
*/
void impLista(Lista l){
if(!esVacia(l)){
impElem(idProceso(l));
impElem(horaLlegada(l));
impElem(duracion(l));
printf("\n");
impLista(resto(l));
}
}

/**
\brief Compara dos datos para saber cual es mayor.
\param e primer dato a comparar.
\param el segundo dato a comparar.
return entero con valor 0 si es falso y cualquier otro
        valor para verdadero.
*/
int EsMoI(Elem e,Elem el){
return e<el;
}

/**
\brief Inserta en orden un proceso segun su hora de llegada
        en una lista de procesos.
\param idP Identificador del proceso.
```

```

\param horaLl Hora de llegada del proceso.
\param dcn Duracion del proceso.
\param l Lista a la que se desea insertar el proceso.
\return Lista con el nuevo proceso insertado
*/
Lista InsOrd(Elem idP, Elem horaLl, Elem dcn, Lista l){
return (esVacia(l))?(cons(idP, horaLl, dcn,vacia())):(
    EsMoI(horaLl,horaLlegada(l)))?
    (cons(idP, horaLl, dcn, l):(cons(idProceso(l), horaLlegada
        (l), duracion(l),
        InsOrd(idP, horaLl, dcn, resto(l)))));
}

```

TDA cola circular:

```

/*
\Libreria del TDA cola
\author: Rojas Zepeda Luis Eduardo
\version 1.0
\date March 31 2019

*/

typedef int Elem;

/**
\brief Estructura que funciona como un nodo de la Cola circular,
    contiene los atributos de cada proceso
    y un apuntador al nodo del siguiente proceso.
*/
typedef struct nodo{
Elem idProceso;
Elem horaLlegada;
Elem duracion;
struct nodo *sig;
}*apNodo;

/**
\brief Estructura que apunta al primer y ultimo elemento de la Cola
    .
*/
typedef struct CNode{
apNodo prim;
apNodo ult;
}*Cola;

/**
\brief Inicializa una Cola vacia.
\return NULL representa el valor vacio de la Cola.
*/
Cola nueva(){
Cola t=(Cola)malloc(sizeof(struct CNode));
t->prim=t->ult=NULL;
return t;
}

/**
\brief Funcion observadora que permite saber si la Cola esta vacia.
\param q Cola que se desea observar.
\return int entero con valor 0 para falso y algo diferente para
    verdadero.
*/
int esNueva(Cola q){

```

```

return (q->prim==NULL)&&(q->ult==NULL);
}

/**
\brief Funcion que permite agregar elementos a la estructura.
\param q Cola a la que se formara el proceso.
\param idProceso Identificador del proceso.
\param horaLlegada Hora de llegada del proceso.
\param duracion Duracion del procesos.
\return q Cola con el proceso agragado.
*/
Cola formar(Cola q, Elem idProceso, Elem horaLlegada, Elem duracion)
{
    apNodo t=(apNodo)malloc(sizeof(struct nodo));
    t->idProceso = idProceso;
    t->horaLlegada = horaLlegada;
    t->duracion = duracion;
    if(esNueva(q)){
        q->prim=q->ult=t;
        t->sig=NULL;
    }else{
        q->ult->sig=t;
        q->ult=t;
        t->sig=NULL;
    }
    return q;
}

/**
\brief Funcion que permite acceder al identificador del primer
        elemento de la Cola.
\param q Cola que se consultara.
\return idProceso Identificador del proceso consultado.
*/
Elem idProcesoC(Cola q){
    return q->prim->idProceso;
}

/**
\brief Funcion que permite acceder a la hora de llegada del primer
        elemento de la Cola.
\param q Cola que se consultara
\return horaLlegada Hora de llegada del proceso consultado.
*/
Elem horaLlegadaC(Cola q){
    return q->prim->horaLlegada;
}

/**
\brief Funcion que permite acceder a la duracion del primer
        elemento de la Cola.
\param q Cola que se consultara.
\return duracion Duracion del proceso consultado.
*/
Elem duracionC(Cola q){
    return q->prim->duracion;
}

/**
\brief Funcion que reduce en una unidad la duracion del primer
        elemento de la Cola.
\param q Cola que se modificara.

```

```
*/
void restaDuracion(Cola q){
q->prim->duracion--;
}

/**
\brief Funcion que desforma el primer elemento de la Cola.
\param q Cola que se modificara.
\return q Cola con el primer elemento retirado.
*/
Cola desformar(Cola q){
if(q->prim==q->ult){
q->prim=q->ult=NULL;
}else{
q->prim=q->prim->sig;
}
return q;
}

/**
\brief Imprime los datos de un proceso.
\param idProceso Identificador que se desea imprimir.
\param duracion Duracion que se desea imprimir.
\param horaLlegada Hora de llegada que se desea imprimir.
*/
void impElemC(Elem idProceso, Elem horaLlegada, Elem duracion){
printf("%d\t\t%d\t\t%d\n",idProceso, horaLlegada, duracion);
}

/**
\brief Imprime una Cola.
\param q Cola que desea imprimir.
*/
void impCola(Cola q){
if(!esNueva(q)){
Cola t=(Cola)malloc(sizeof(struct CNode));
impElemC(idProcesoC(q), horaLlegadaC(q), duracionC(q));
t->prim=q->prim;
t->ult=q->ult;
impCola(desformar(t));
}
}

Cola rotar(Cola q){
if(esNueva(q)){
return q;
}else{
return formar(desformar(q),idProcesoC(q), horaLlegadaC(q),
duracionC(q));
}
}
```

Archivo Makefile

```
1 programa: Src/programa.c
2     $(CC) Src/programa.c -o Src/progra.out
3
```

Y finalmente crearemos el archivo README.txt con el siguiente texto:

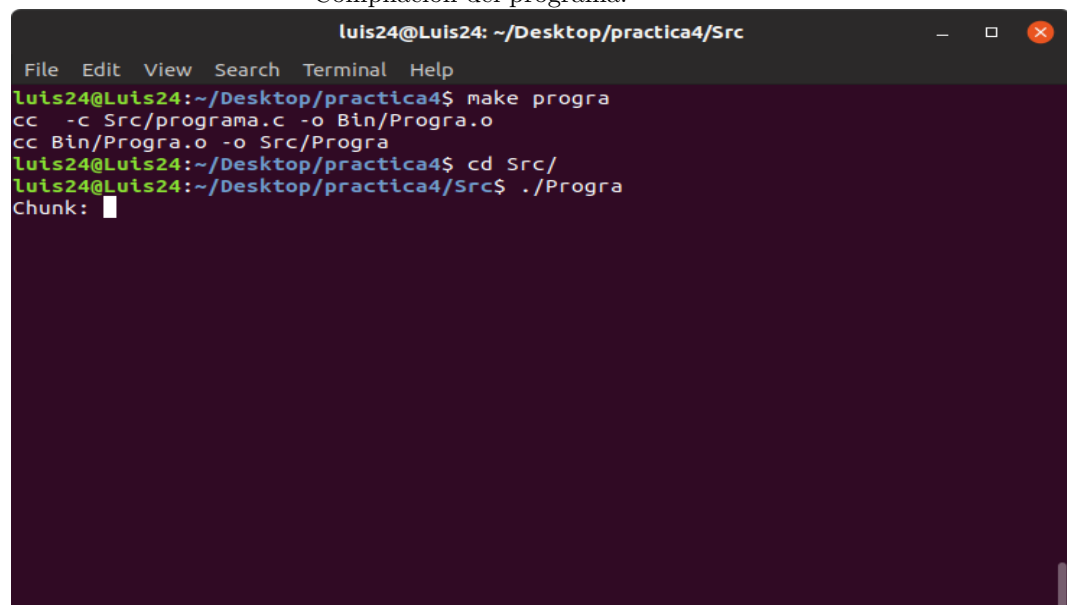
Archivo README

```
1 COMPILACION DEL PROGRAMA:
2     make programa
3
4 EJECUCION:
5     posicionarse en el la carpeta Src y ejecutar:
6     ./prgra
7
```

Este archivo servirá como manual para el usuario para correr los programas de manera más fácil. Este archivo, junto con el makefile, irán en la raíz de la carpeta como se muestra en la primera imagen de esta sección de desarrollo.

Posteriormente para compilar el programa se hará de la siguiente forma:

Compilación del programa:

A screenshot of a terminal window titled 'luis24@Luis24: ~/Desktop/practica4/Src'. The terminal shows the following commands and output:

```
luis24@Luis24:~/Desktop/practica4$ make progra
cc -c Src/programa.c -o Bin/Progra.o
cc Bin/Progra.o -o Src/Progra
luis24@Luis24:~/Desktop/practica4$ cd Src/
luis24@Luis24:~/Desktop/practica4/Src$ ./Progra
Chunk: 
```

Y se verá de la siguiente forma la ejecución:

```
luis24@Luis24: ~/Desktop/practica4/Src
File Edit View Search Terminal Help
luis24@Luis24:~/Desktop/practica4/Src$ ./a.out
Chunk: 5

2 Aqui estoy (se formo este preceso)

IdProceso      HoraLlegada      Duracion
2              0                6

Soy el proceso 2
Soy el proceso 2

1 Aqui estoy (se formo este preceso)

IdProceso      HoraLlegada      Duracion
2              0                4
1              2                3

Soy el proceso 2
Soy el proceso 2
Soy el proceso 2

3 Aqui estoy (se formo este preceso)

IdProceso      HoraLlegada      Duracion
2              0                1
1              2                3
3              5                4

4 Aqui estoy (se formo este preceso)

IdProceso      HoraLlegada      Duracion
1              2                3
3              5                4
2              0                1
4              6                4

Soy el proceso 1
```



```
luis24@Luis24: ~/Desktop/practica4/Src
File Edit View Search Terminal Help
1          2          3
3          5          4

4 Aqui estoy (se formo este preceso)

IdProceso    HoraLLegada    Duracion
1            2            3
3            5            4
2            0            1
4            6            4

Soy el proceso 1
Soy el proceso 1
Soy el proceso 1

1 Adios

Soy el proceso 3
Soy el proceso 2

2 Adios

Soy el proceso 4
Soy el proceso 4
Soy el proceso 4
Soy el proceso 3
Soy el proceso 3
Soy el proceso 3

3 Adios

Soy el proceso 4

4 Adios

luis24@Luis24:~/Desktop/practica4/Src$
```

4. Bibliografía

Referencias

- [1] <https://es.wikipedia.org/wiki/Planificaci>
- [2] <http://blog.furiosojack.com/2017/04/explicacion-de-round-robin.html>
- [3] <https://users.dcc.uchile.cl/~bebustos/apuntes/cc30a/TDA/>