TP I – Planificador de Procesos

Objetivo: Desarrollar un programa que simule la planificación de procesos que realiza un SO que trabaja según el modelo de procesos de dos estados.

Características generales del programa a realizar:

- Implemente el programa en el lenguaje C, utilizando el compilador GNU GCC, puede utilizarlo directamente desde cualquier distribución Linux que lo tenga instalado o bien desde el port GCC para Windows cygwin (http://www.cygwin.com)
- El programa puede trabajar con un número fijo y predeterminado de procesos a planificar
- El programa debe emitir la traza de los procesos que va ejecutando
- Dar una cuota o quantum de ejecución de n instrucciones para cada proceso
- Bloquear procesos que requieran operaciones de I/O y pasar el control a otro proceso que este en estado "Not Running"
- El programa termina su ejecución cuando ya se han terminado todos los procesos
- La cola de procesos puede simularse con un arreglo

Sugerencias, posible implementación:

Se pueden definir un conjunto de posibles instrucciones a ejecutar por cada proceso:

```
// intrucciones posibles
#define ICALC       0    // calculo
#define IPUTMEM     1    // poner un valor en memoria
#define IGETMEM     2    // recuperar un valor de memoria
#define ILOGIC      3    // operacion logica
#define IIO         4    // operacion de I/O bloqueante
#define IEND        99   // finalizacion de programa
```

Se pueden definir programas, como un conjunto de instrucciones:

```
// programas
#define PROG_1      ICALC, ICALC, ILOGIC, IPUTMEM, IIO, ICALC, ICALC, IEND
#define PROG_2      ICALC, ILOGIC, ILOGIC, ICALC, IGETMEM, ICALC, ICALC, IEND
#define PROG_3      ICALC, ICALC, ILOGIC, IPUTMEM, ICALC, ICALC, ICALC, IEND
#define PROG_4      ICALC, ILOGIC, ILOGIC, IPUTMEM, IGETMEM, ICALC, ICALC, IEND
#define PROG_5      ICALC, ICALC, ILOGIC, IPUTMEM, IGETMEM, IPUTMEM, ILOGIC, ILOGIC, IEND
```

Se pueden definir un conjunto de MAX_PROC procesos a gestionar, cada uno con máximo de PROGRAM_LEN instrucciones cada uno:

```
int progs[MAX_PROC][PROGRAM_LEN] = { {PROG_1}, {PROG_2},{PROG_3},{PROG_4},{PROG_5}, };
```

Se puede definir un bloque de control de proceso (Process Control Block) por cada proceso a gestionar, que guarde además del programa (code), el id de proceso, el estado y el contador de programa (program counter):

```
typedef struct {
   int id;
   int state;
   int pc;
   int code[PROGRAM_LEN];
} pcb;
```

y simular la cola de procesos con un arreglo de MAX_PROC procesos a gestionar:

```
pcb cola[MAX_PROC];
```

podríamos también definir los posibles estados de los procesos a gestionar:

```
#define RUNNING     1
#define NOT_RUNNING 0
```

El arreglo `cola[]` requerirá una inicialización a estado "Not Running", pc (program counter) = 0, generar un id de proceso, copiar las instrucciones de cada programa al arreglo code. Todo esto lo podemos encapsular en la función `pcbinit()`:

```
void pcbinit() {
   int n;
   for(n=0;n<MAX_PROC;n++) {
      cola[n].id=n+100;
      cola[n].state=NOT_RUNNING;
      cola[n].pc=0;
      // cargo instrucciones de procesos
      memcpy(&cola[n].code,&progs[n][0],sizeof(int)*PROGRAM_LEN);
   }
}
```

También sería útil (para debug y testing) contar con una función para la impresión por pantalla de la cola de procesos y otra función que encapsule todo el trabajo del planificador, por lo tanto, nuestro programa principal podría tener la forma:

```
// programa principal
int main() {
   pcbinit();      // inicializo cola de procesos
   pcbprint();     // imprimo cola de procesos
   run();          // ejecuto planificador de procesos
   pcbprint();     // imprimo cola de procesos
   return 0;       // fin
}
// fin programa principal
```

Ahora queda pensar la implementación de la función `run()` acorde con el modelo de dos estados visto en clase. Discuta con sus compañeros y profesor alguna implementación posible.