Escalonamento Posix 1003.1b

Arquivo fonte: posix1003.c, posix1003.cc ou posix1003.cpp

1. Tarefa

Este trabalho consiste na implementação de um simulador para teste do algoritmo de escalonamento definido no padrão Posix (*Portable Operating System Interface*) 1003.1b, aplicado a um conjunto de tarefas aperiódicas.

O padrão Posix 1003.1b utiliza escalonamento preemtivo com prioridades fixas, com pelo menos 32 níveis de prioridade. Nesse padrão, o escalonamento funciona em dois níveis: (1) seleção da fila de prioridade mais alta em que haja pelo menos uma tarefa (*thread* ou processo) pronta; (2) seleção de uma tarefa dessa fila de acordo com a sua política de escalonamento.

São definidas 3 políticas de escalonamento: SCHED_FIFO, SCHED_RR e SCHED_OTHER. E cada tarefa pode ou herdar uma política ou definí-la explicitamente.

Na política SCHED_FIFO (*First-In-First-Out*), quando uma tarefa entra no estado "pronta" (*ready*) ela é inserida no final da fila de tarefas do seu respectivo nível de prioridade. Se este for o nível de prioridade mais alto que tiver pelo menos uma tarefa pronta, o processador será atribuído à primeira tarefa desta lista. Essa tarefa poderá liberar o processador quando for bloqueada, quando terminar a execução ou quando for preemptada, por exemplo, devido à chegada de uma tarefa com maior nível de prioridade.

A política SCHED_RR (*Round-Robin*), também respeita a ordem de chegada das tarefas, mas estabelece um limite de tempo máximo de uso do processador (*quantum* ou *time-slice*) para cada tarefa. Depois deste limite de tempo, a tarefa é preemptada, inserida no final fila e o processador é passado para a próxima tarefa desse mesmo nível de prioridade.

A política SCHED_OTHER permite o uso de uma política implementada. Neste trabalho, esta política NÃO será considerada.

Em seus materiais de aula, Singhoff (2013) apresenta um exemplo de escalonamento segundo o padrão Posix 1003.1b. Considerando os seguintes processos:

Tarefa	Computação (C_i)	Chegada (S_i)	Prioridade (p_i)	Política
A	1	7	1	SCHED_FIFO
B	5	0	4	SCHED_RR
C	3	0	4	SCHED_RR
D	6	4	2	SCHED_FIFO

E também considerando uma fatia de tempo de 1 unidade para SCHED_RR, seria produzida a seguinte ordem de escalonamento:

BCBCDDDADDDBCBB

O simulador a ser implementado neste trabalho deverá: ler da entrada padrão um conjunto de valores que correspondem à definição de um conjunto de tarefas (número de tarefas e para cada tarefa o tempo de computação, o tempo de chegada, a prioridade e a política de escalonamento), conforme definido na Seção 2 (Entrada); e gerar na saída o resultado da simulação (ordem ou grade de execução) para cada conjunto de tarefas, confome definido na Seção 3 (Saída).

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um número inteiro *N*, que indica o número de tarefas. A seguir aparecem na entrada as descrições de cada uma das *N* tarefas. A descrição de cada tarefa é composta por três valores que correspondem respectivamente a: tempo de computação da tarefa (C_i), tempo de chegada da tarefa (S_i) e prioridade

(p_i) da tarefa (valor de 1 a 32, com 1 correspondendo à maior prioridade e 32, à menor prioridade) e política de escalonamento (SCHED_FIFO correspondendo a 1 e SCHED_RR, a 2). O final das entradas é indicado por $N = 0$.

Os valores de entrada devem ser lidos da entrada padrão (normalmente o teclado) – por exemplo, com *scanf()* ou *getchar()*, em C –, de forma que seja possível redirecionar um arquivo para o processo. Não se deve utilizar arquivos de entrada, nem funções para esperar pelo pressionamento de teclas (comuns quando se depura um programa).

Exemplo de Entrada

```
4
1 7 1 1
5 0 4 2
3 0 4 2
6 4 2 1

0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve executar a simulação da execução das tarefas usando o algoritmo de escalonamento do padrão Posix 1003.1b.

A simulação deve ser apresentada em uma única linha, mostrando uma simplificação do diagrama de Gantt correspondente à execução dos processos (tal como apresentado no exemplo na seção inicial deste documento). Nesta linha, usam-se caracteres para representar unidades de execução no processador. Uma unidade de execução corresponde à execução da primeira tarefa é indicada pelo caractere 'A'. Uma unidade de execução da segunda tarefa, pelo caractere 'B'. E assim sucessivamente. Unidades de execução ociosas são indicadas pelo caractere '.' (ponto).

Como a entrada pode ser composta por vários conjuntos de teste, o resultado de cada conjunto deverá ser separado por uma linha em branco. A grafia mostrada no Exemplo de Saída, a seguir, que corresponde ao resultado esperado para o exemplo de entrada apresentado anteriormente, deve ser seguida rigorosamente.

Os valores de saída devem ser escritos na saída padrão (normalmente o vídeo) – por exemplo, com *printf()* ou *putchar()*, em C –, de forma que seja possível redirecionar a saída gerada pelo processo para um arquivo texto qualquer. Não se deve utilizar arquivos de saída, nem funções para limpar a tela.

Exemplo de Saída

```
BCBCDDDADDDBCBB
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$1 <= N <= 26$ ($N = 0$ apenas para indicar o final da entrada)

$1 <= C_i <= 2048$

$1 <= T_i <= 2048$

$1 <= p_i <= 32$

Referências

SINGHOFF, Frank. **Programming Real-Time Embedded systems**: C/POSIX and RTEMS. Brest, France: University of Brest, 2013. Lecture Materials. Disponível em: <http://beru.univ-brest.fr/~singhoff/ENS/USTH/posix.pdf>. Acesso em 31 ago. 2017.

Autor: Roland Teodorowitsch