MC202 — ESTRUTURAS DE DADOS

Lab 06 — Fila de prioridade

Problema

Killua está desenvolvendo um Sistema Operacional simples, contudo um Sistema Operacional tem muitas funcionalidades e ele precisa de ajuda para desenvolvê-las. Uma dessas funcionalidades é o *Escalonador de Processos* e Killua ainda não possui um Escalonador de Processos para seu sistema operacional. Um Escalonador de Processos é um subsistema do Sistema Operacional responsável por decidir o momento em que cada processo usará a CPU[1]. Para o Sistema Operacional que o Killua está montando, ele precisa de um Escalonador de Processos que opere sobre as seguintes condições:

- Dados os processos e suas prioridades, o escalonador deve selecionar os processos com as maiores prioridades para serem executados pelos N núcleos do processador.
- Que o escalonador selecione sempre um novo processo com a maior prioridade (em caso de empate, o de menor identificador) para ser executado assim que algum núcleo do processador fique livre. Caso haja mais de núcleo livre, atribua o processo ao núcleo de menor identificador. Um processo é encerrado quando se passarem os E *clocks* necessários para executar o processo.
- O escalonador deve garantir que um processo nunca é executado antes de outro processo do qual depende. Um processo tem no máximo uma dependência e não existem dependências cíclicas entre eles, por exemplo o processo 1 depende do 2 e o 2 depende do 1.

Killua possui muita dificuldade em implementar filas de prioridades, por isso ele pediu sua ajuda para implementar o Escalonador de Processos para o Sistema Operacional que ele está desenvolvendo.

Entrada

A primeira linha contém dois inteiros, $N \geq 1$ e $P \geq 1$, que indicam o número de núcleos disponíveis e o número de processos. Em seguida são dadas P linhas, uma para cada processo contendo: o identificador do processo I; as quantidades de *clocks* $E > 0$ necessários para executar o processo; a prioridade do processo H; e o identificador D da dependência. D igual a 0 indica que um processo não tem dependência.

[1] https://www.oficinadanet.com.br/post/12781-sistemas-operacionais-o-que-e-escalonamento-de-processos

Saída

A saída do seu programa deve exibir $2P$ linhas, uma linha para quando o processo foi iniciado e outra para quando o processo foi encerrado. A linha que indica o início do processo deve seguir o seguinte formato: "`processo <id> iniciou no clock <clock>`", onde `<id>` e `<clock>` deve ser substituído pelo identificador e o *clock* em que o processo iniciou a execução (iniciando de 0), respectivamente. Caso mais de um processo tenha sido selecionado para executar no mesmo *clock*, então deve-se exibir primeiro aquele que tem maior prioridade e, em caso de empate, o que possui menor identificador. A linha de processo encerrado deve seguir o seguinte formato: "`processo <id> encerrou no clock <clock>`". Caso mais de um processo encerre ao mesmo tempo, deve-se imprimir pela ordem dos núcleos. Por exemplo, se um processo **A** está executando no núcleo 1 e o processo **B** no núcleo 0, e ambos encerram ao mesmo tempo, o processo **B** deve ser impresso antes do **A**. Em um dado *clock* a impressão de encerrar deve ser feita antes da impressão de iniciar.

Exemplo

Entrada

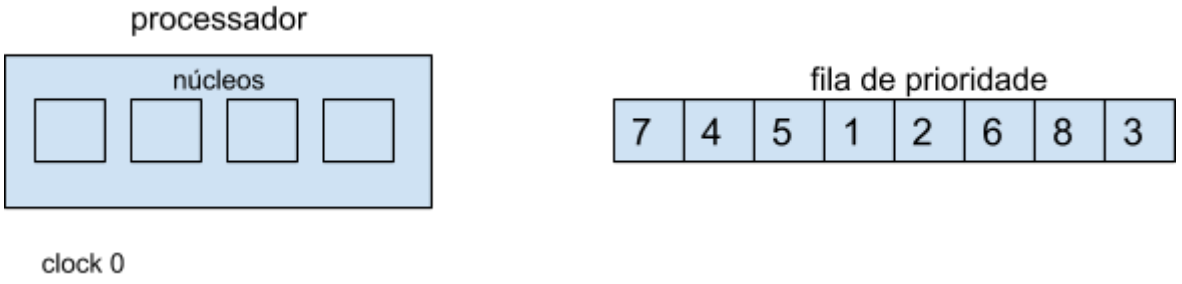
```
4 8
1 6  10 7
2 9  9  1
3 12 3  0
4 15 12 0
5 18 12 0
6 21 7  2
7 24 14 0
8 27 5  0
```

Saída

```
processo 7 iniciou no clock 0
processo 4 iniciou no clock 0
processo 5 iniciou no clock 0
processo 8 iniciou no clock 0
processo 4 encerrou no clock 15
processo 3 iniciou no clock 15
processo 5 encerrou no clock 18
processo 7 encerrou no clock 24
processo 1 iniciou no clock 24
processo 3 encerrou no clock 27
processo 8 encerrou no clock 27
processo 1 encerrou no clock 30
processo 2 iniciou no clock 30
```

```
processo 2 encerrou no clock 39
processo 6 iniciou no clock 39
processo 6 encerrou no clock 60
```

Inicialmente os núcleos estão livres e os processos são escalonados



Em seguida, 4 processos são selecionados para executar nos núcleos.

No clock 0 são selecionados os processo 7, 4, 5 e 8, que são os processos de maior prioridade e sem dependência (note que os processos 1, 2 e 6 tem prioridade maior que o processo 8, contudo eles possuem dependências que ainda não foram atendidas), para serem executados pelos 4 núcleos.

No clock 15, que é o tempo necessário para encerrar o primeiro processo que está em execução, o processo 4 é encerrado.

O core 2 fica livre e novamente um processo é selecionado para ser executado.

clock 15

O processo 3 é selecionada para executar no core 2, note novamente que 1, 2 e 6 tem prioridades maiores, contudo suas dependências ainda não foram atendidas.
No clock 18 o processo 5 encerra.

clock 18

Como os processo 1, 2 e 6 possuem dependências ainda não atendidas eles não podem executar e o core 3 fica livre.
No clock 24 o processo 7 encerra.

clock 24

Agora temos dois núcleos livre e novamente iremos selecionar os processos para executar, porém 2 e 6 ainda tem dependência e apenas o processo 1 pode ser executado.

Agora que o processo 1 pode ser executado ele é atribuído ao núcleo 1, que é o núcleo disponível com menor identificador.
Este processo continua até todos os processos serem executados.

Este exemplo é meramente ilustrativo, sua função é apenas para auxiliar os alunos que podem encontrar dificuldades em entender o que o lab está pedindo. Sendo assim, suas implementações não precisam necessariamente seguir a ideia dessa ilustração, contando que os critérios de correção sejam satisfeitos.

Observações

- Neste laboratório, é obrigatório implementar e utilizar fila de prioridade.
- Para as turmas A, B e C, esse laboratório tem peso 2.
- Deverão ser submetidos os seguintes arquivos: **`lab06.c`**, **`fila_prioridade.c`** e **`fila_prioridade.h`**.
- Observações sobre SuSy:
    - Versão do GCC: 4.4.7 20120313 (Red Hat 4.4.7-17).
    - *Flags* de compilação:
        - `-ansi -Wall -pedantic-errors -Werror -g -lm`
    - Utilize comentários do tipo `/* comentário */`;
      comentários do tipo `//` serão tratados como erros pelo SuSy
    - Tempo máximo de execução: 1 segundos.
    - Evite usar o sistema como compilador ou apenas para testar as instâncias.
    - Sempre use o software valgrind para detectar erros decorrentes do uso incorreto da memória dinâmica **em todos os testes abertos**. Relembrando:
        - `valgrind --leak-check=full ./lab06 < arq01.in`
- Além das observações acima, esse laboratório será avaliado pelos critérios:
    - Indentação de código e outras boas práticas, tais como:
        - uso de comentários (apenas quando são relevantes);
        - código simples e fácil de entender;
        - sem duplicidade (partes que fazem a mesma coisa).
    - Organização do código:
        - arquivo de cabeçalho (.h) contendo definições de constantes, tipos de dados criados pelo usuário, protótipo de funções etc;

        - arquivos de implementação (.c) que implementam as funções definidas no arquivo de cabeçalho;
        - tipos de dados criados pelo usuário e funções bem definidas e tão independentes quanto possível.
    - Corretude do programa:
        - programa correto e implementado conforme solicitado no enunciado;
        - desalocar toda memória alocada dinamicamente durante a execução do programa;
        - não realizar leitura ou escrita em blocos de memória não alocados;
        - inicialização de variáveis sempre que for necessário;
        - dentre outros critérios.