Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Sistemas Operacionais – 2020/2

Trabalho Prático #2 – Escalonador de Processos

1) Introdução

Ao longo desse trabalho você utilizará um sistema operacional simples de aprendizado chamado XV6. O XV6 é simples o bastante para que seja possível entende-lo e modifica-lo em poucas semanas, e, ao mesmo tempo, abrange os conceitos mais importantes da estrutura do UNIX. Para executá-lo, você precisará compilar os arquivos fontes e usar o emulador de processador QEMU.

O XV6 é desenvolvido como parte da disciplina de Sistemas Operacionais do MIT. Você poderá encontrar informações úteis no link oficial.

Dica: O XV6 também tem um manual bastante útil. Esse manual irá te ajudar muito ao longo desse trabalho.

2) Obtendo o XV6

Para implementação deste trabalho, você precisará baixar o XV6 a partir do repositório no GitHub do MIT. Para isso, siga as etapas a seguir:

1. Abra o **`shell`** no seu sistema operacional, crie uma pasta para o trabalho, por exemplo, **`tp2`** e acesse essa pasta:

```
$ mkdir ~/tp2
$ cd tp2
```

2. Execute o comando abaixo para baixar o XV6 na pasta **`xv6-public`**:

```
$ git clone git://github.com/mit-pdos/xv6-public.git
```

3. Execute o comando a seguir para instalar o QEMU:

```
$ sudo apt install qemu-system-x86
```

4. Agora, acesse a pasta xv6-public e compile o XV6 com o comando **`make`**:

```
$ cd xv6-public
$ make
```

5. Para rodar o XV6 dentro do QEMU execute o comando abaixo:

```
$ make qemu-nox
```

Esse comando irá compilar o QEMU e vai iniciar automaticamente o XV6 dentro do QEMU em seu terminal. Para sair, você poderá fechar a janela, digitar **`Ctrl+A X`** ou abrir o monitor do QEMU utilizando o comando **`Ctrl+A C`** e digitar o comando **`quit`**.

3) Escalonador de processos

O escalonador é um dos componentes mais básicos e importantes de um sistema operacional. O escalonador deve satisfazer vários objetivos conflitantes: tempo de processamento rápido, boa vazão para tarefas que rodam em segundo plano, evitar a inanição de processos, conciliar as necessidades de processos de alta a baixa prioridades, etc.

O conjunto de regras usados para determinar quando e como selecionar um novo processo para executar é chamado de *política de escalonamento*.

Para cumprir sua primeira tarefa você deve entender a política atual de escalonamento do XV6. Para isso, você deve ler o capítulo 5 do manual do XV6. Em seguida, localize o código do escalonador no código fonte do XV6 e entenda o seu fluxo de execução. Tente responder as seguintes questões para auxiliar em seu entendimento:

- Qual processo a política atual de escalonamento seleciona para rodar?
- O que acontece quando um processo retorna de uma tarefa de I/O?
- O que acontece quando um processo é criado e quando ou quão frequente o escalonamento acontece?

Sua primeira tarefa é modificar a política atual do escalonador para que o processo de preempção ocorra a cada intervalo *n* de tempo (medidos em *ticks* do clock) ao invés de a cada 1 *tick* do clock. Para isso, adicione a linha seguinte ao arquivo **`param.h`** e inicialize o valor `INTERV` para 5. Utilize essa constante para estabelecer o intervalo entre preempções.

```
#define INTERV 5
```

4) Escalonamento de Filas Multinível

O escalonamento de filas multinível (*Multi-Level Queue Scheduling*) é uma política de escalonamento preemptiva que inclui 3 filas de prioridades. Inicialmente, cada processo deve ser iniciado com prioridade padrão 2 e essa prioridade deve ser atribuída a partir da chamada **`fork`**. Nessa política de escalonamento, o escalonador irá selecionar um processo de uma fila de baixa prioridade somente se não houver processo pronto para rodar em uma fila de prioridade mais alta. A prioridade 2 é a mais alta, a prioridade 1 é a intermediária e a prioridade 0 é a mais baixa. A mudança de um processo entre filas de prioridades é somente possível via *system call*. Para isso, você deverá criar a chamada de sistema **`set_prio`** que altera a prioridade atual do processo.

```
int set_prio(int priority)
```

O parâmetro `priority` será um número de inteiro do conjunto {0, 1, 2} indicando a nova prioridade do processo. Essa função retorna `0` se bem sucedido ou `-1` se houver um erro.

Dica: para aprender como criar uma chamada de sistema consulte este tutorial.

Sua nova tarefa é modificar o atual escalonador do XV6 para implementar uma política de escalonamento de filas multinível como descrito acima. Posteriormente, você deverá adicionar uma mecanismos de *aging* para evitar inanição. Dessa forma, você deve criar mais duas constantes para com a informação do tempo de espera de um processo antes que ele seja realocado para a fila como maior prioridade. Por exemplo `0TO1` e `1TO2`.

5) Testes

Para analisar o efeito da política implementada você deve extrair algumas estatísticas de cada processo executado. Para isso você deverá estender a estrutura `proc` no arquivo **`proc.h`**. Adicione à estrutura `proc` os seguintes campos:

```
uint ctime; // Tempo quando o processo foi criado
int stime; //Tempo SLEEPING
int retime; //Tempo READY(RUNNABLE) time
int rutime; // Tempo executando (RUNNING)
```

Esses campos representam respectivamente o tempo da criação do processo, o tempo gasto no estado `SLEEPING`, o tempo gasto no estado `READY` e tempo gasto executando (`RUNNING`).

A partir da criação do processo o *kernel* deverá atualizar o campo `ctime`. Os outros campos deverão ser atualizados a cada *tick* do clock. Você pode assumir que um processo está no estado `SLEEPING` somente quando estiver esperando uma tarefa de I/O. Tome cuidado ao marcar o tempo de terminação de um processo: note que um processo pode ficar um tempo arbitrário no estado `ZOMBIE`.

Para extrair essas informações de cada processo e apresentá-la para o usuário você também deverá criar outra chamada de sistema. Essa nova chamada de sistema `wait2` deverá estender a chamada de sistema atual `wait`:

```
int wait2(int* retime, int* rutime, int* stime)
```

Essa chamada irá atribuir ao parâmetro `retime` o tempo em que o processo esteve no estado `READY`, irá atribui ao parâmetro `rutime` o tempo em que o processe esteve no estado `RUNNING` e ao parâmetro `stime` o tempo em que o processo esteve no estado `SLEEPING`. Deverá ainda retornar `0` se for bem sucedido ou `-1` se houver erro.

Uma vez criada essa chamada de sistema você deverá criar um programa chamado **`sanity.c`** que recebe como argumento um parâmetro inteiro *n*. Esse programa irá criar 5 vezes *n* processos

com `fork` e esperar até que cada um deles termine imprimindo as estatísticas da chamada de sistema `wait2` para cada processo terminado.

Cada um dos 5*n* processos será de um dos 3 tipos abaixo:

- Processos com (`pid mod 3 == 0`) serão processos do tipo CPU-Bound: executam 100 vezes um *loop* vazio de 1.000.000 iterações.
- Processos com (`pid mod 3 == 1`) serão processos de tarefas curtas S-CPU: executam 100 vezes um *loop* vazio de 1.000.000 iterações e a cada 100 iterações chama a função do sistema `yield`.
- Processos com (`pid mod 3 == 2`) serão processos IO-Bound: para simular chamadas de IO executa 100 vezes a chamada de sistema `sleep(1).`

Esse programa deverá ainda imprimir para cada processo terminado:

- O identificador do processo (pid) e o seu tipo {CPU-Bound, S-Bound, IO-Bound}.
- O seu tempo de espera, tempo executando e tempo de IO.

Após todos os *5n* processo serem executados imprima:

- Tempo médio no estado `SLEEPING` para cada tipo de processo.
- Tempo médio READY para tipo de processo.
- Tempo médio para completar para completar (*turnaround time*) a tarefa para tipo de processo.

6) Entrega

A data de entrega é 07/10/2020, quarta-feira, até as 23:55 via *moodle*.

Esse trabalho poderá ser feito em dupla. Seu grupo deverá submeter no *moodle* um único arquivo no formato .zip contendo o código fonte do XV6 (pasta `xv6-public`) e um relatório (.pdf) contendo nomes dos componentes do grupo e explicando os algoritmos implementados e a análise dos resultados dos testes obtidos.

7) Esclarecimentos

1. **Leia o manual do XV6!** Principalmente o capítulo 5. O entendimento desse capítulo irá ajudar bastante você a entender o funcionamento do escalonador do XV6 e como modificá-lo para esse trabalho.
2. Serão disponibilizados códigos auxiliares para que vocês possam se basear para implementação da carga de testes proposta no trabalho prático e verificação das políticas de escalonamento requisitadas.