KernelSim – INF1316 – Trabalho 1

É um simulador de núcleo de sistema operacional com escalonamento preemptivo round-robin (RR) e dispositivo de entrada e saída simulado (D1), usando processos unix, sinais, memória compartilhada (SHM) e FIFO.

Integrantes do Grupo

Miguel Mendes - 2111705

Igor Lemos - 2011287

Estrutura

- `kernel` —> KernelSim: faz RR com quantum configurável e preempção com SIGSTOP/SIGCONT, além de bloqueio e desbloqueio por I/O (IRQ1), e coordena SHM e FIFO;
- `inter_controller` —> InterController Sim: gera as seguintes interrupções:
  - IRQ0 (SIGUSR1) a cada 1s. Fim do time-slice;
  - IRQ1 (SIGUSR2) 3s após cada pedido de I/O.
- Aplicações que fizemos para teste (Ai):
  - `app_rw` —> pede I/O em `pc=3` (READ) e `pc=8` (WRITE), alternando R/W;
  - `app_cpu` —> não pede I/O (CPU apenas), é bom só para ver a preempção "pura".

Funcionamento

- RR com quantum de 1 segundo (padrão): a cada IRQ0, o kernel preempta quem está na CPU (SIGSTOP) e despacha o próximo pronto (SIGCONT);
- Quando detecta `want_io[idx]`, o kernel bloqueia o processo (ST_WAITING), enfileira o pedido em FIFO (`PID TIPO`) e tira ele da CPU;
- O `inter_controller` lê o FIFO, atende 1 pedido por vez por 3s e, ao concluir, preenche `io_done_pid/type` na SHM e envia IRQ1 (SIGUSR2);
- No IRQ1, o kernel desbloqueia o processo correspondente e dá prioridade: preempta a CPU e despacha o desbloqueado;
- O PC de cada Ai é salvo na SHM e, ao receber `SIGCONT`, o processo restaura seu PC e continua de onde parou garantindo que execute exatamente `MAX` iterações.

Build

```
gcc -Wall -o kernel kernel.c
gcc -Wall -o inter_controller inter_controller.c
gcc -Wall -o app_rw app_rw.c
gcc -Wall -o app_cpu app_cpu.c
```

O kernel atualmente executa sempre `./app` para cada Ai. Para você escolher qual app testar, crie um link simbólico:

```
ln -sf app_rw app      # testar app_rw
# ou
ln -sf app_cpu app     # testar app_cpu
```

Formato para rodar:

```
./kernel <quantum_s> <duracao_s> -- ./app [-- ./app]...
```

Exemplos: - 3 processos app_rw:

```
ln -sf app_rw app
./kernel 1 25 -- ./app -- ./app -- ./app
```

- 4 processos app_cpu:

```
ln -sf app_cpu app
./kernel 1 25 -- ./app -- ./app -- ./app -- ./app
```

- Dá para ajustar o número de processos entre 3 e 6 repetindo `-- ./app`. Para ver os apps chegarem no limite que colocamos de 20 iterações, use duração maior (> 40s).

Linha do Tempo

Cenário: 3 apps `app_rw`, quantum=1s, I/O de 3s.

1. `t=0s` — Kernel inicia, despacha `A1` e o InterController loga `INÍCIO`;
2. `t=1s` — IRQ0: Kernel preempta `A1` e despacha `A2`;
3. `t=2s` — IRQ0: Kernel preempta `A2` e despacha `A3`;
4. `t 3s` — `A1` chega em `pc=3`, pede I/O READ. Kernel bloqueia `A1` e enfileira. InterController lê FIFO e inicia atendimento de `A1` (3s);
5. `t=4s` e `t=5s` — RR continua entre os prontos. `A2` e `A3` executam;
6. `t 6s` — InterController conclui I/O de `A1` e envia IRQ1. Kernel desbloqueia `A1` e despacha com prioridade;
7. `t 8-9s` — `A1` pede WRITE em `pc=8`. Ciclo repete (bloqueio, atendimento por 3s, IRQ1, desbloqueio com prioridade);
8. Fila de I/O: se mais processos pedirem I/O, são atendidos um por vez, 3s cada, na ordem de chegada.

Limitações

- O kernel executa sempre `./app` para cada processo. Para testar `app_rw` e `app_cpu` na mesma execução, talvez seria melhor adaptar o `spawn_apps` do kernel para usar os caminhos passados após `--`.