EP1

MAC0422 - Sistemas Operacionais (2020)

bccsh.c

O shell desenvolvido foi capaz de permitir a invocação externa dos 3 binários:

- /usr/bin/du -hs
- /usr/bin/traceroute www.google.com.br
- ./ep1 <argumentos do EP1>

Além de mkdir, kill e ln, todos esses comandos que já são embutidos internamente do linux por execve, assim sendo abrasileirados com nomes de fazdir, homicidio e liga, respectivamente.

bccsh.c

O shell espera um comando com read_input(entrada, prompt), assim

```
1.  if (read_input(entrada, prompt))
2.          continue;
```

O parse_input trabalha com a string de input do comando, se os argumentos passarem do tamanho máximo irá dar exit(1), assim separando os comandos.

```
1.  pid_t pid = fork();
2.  if (pid != 0)
3.          wait(NULL);       //Quando for o pai, espere terminar de rodar o filho
4.   else {
5.        execcommand(args);  //args sendo os comandos
6.        exit(0); }
```

bccsh.c

execcommand é quem lida com os comandos internos, se é embutido como o mkdir, kill e ln ou um binário.

A função func_id(argv[0]) irá retornar se for algum index dos comandos internos char* embutidas[] = { "mkdir", "kill", "ln", "quit" }; se não for algum desses irá rodar a função execve, para programas como /usr/bin/du -hs, /usr/bin/traceroute ou o ./ep1

ep1.c

O ep começa com uma criação de fila de processos:

Para lidar com os processos, implementamos uma Fila Circular, assim sendo bem mais fácil trabalhar com os escalonadores. Essa fila utiliza do Trace que é o ponteiro da estrutura que usamos para armazenar os detalhes dos processos

- struct trace {
  - char* nome;
  - int t0;
  - long dt;
  - int deadline;
  - long remaining;
  - long nremaining;
  - int id;
- }

ep1.c

Com essa fila de processos, podemos começar por padrão o escalonador_init(), essa função tem como entrada a Fila e aloca na memoria ram um array de p_threads do tamanho da Fila, além disso trabalha com o mutex e o cond, assim retornando o array de p_threads.

Todos os escalonadores chama a função para conseguir trabalhar com as threads na CPU.

Com isso podemos explicar cada escalonador

Escalonadores

Os escalonadores são implementados com uma chamada de thread própria para o controle dos processos.

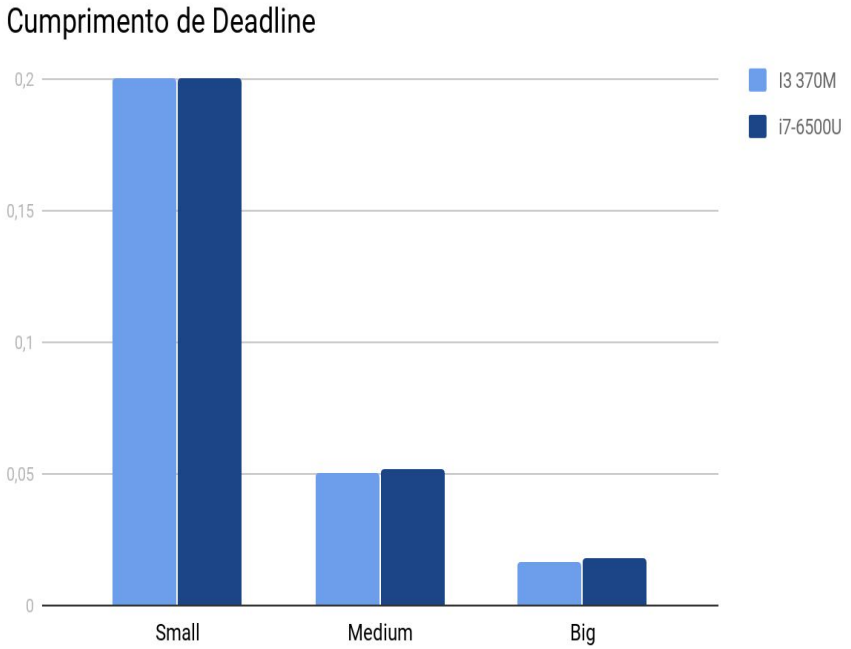
Por questões de simplicidade os escalonadores presumem apenas um cpu por onde os processo podem ter acesso aos recursos

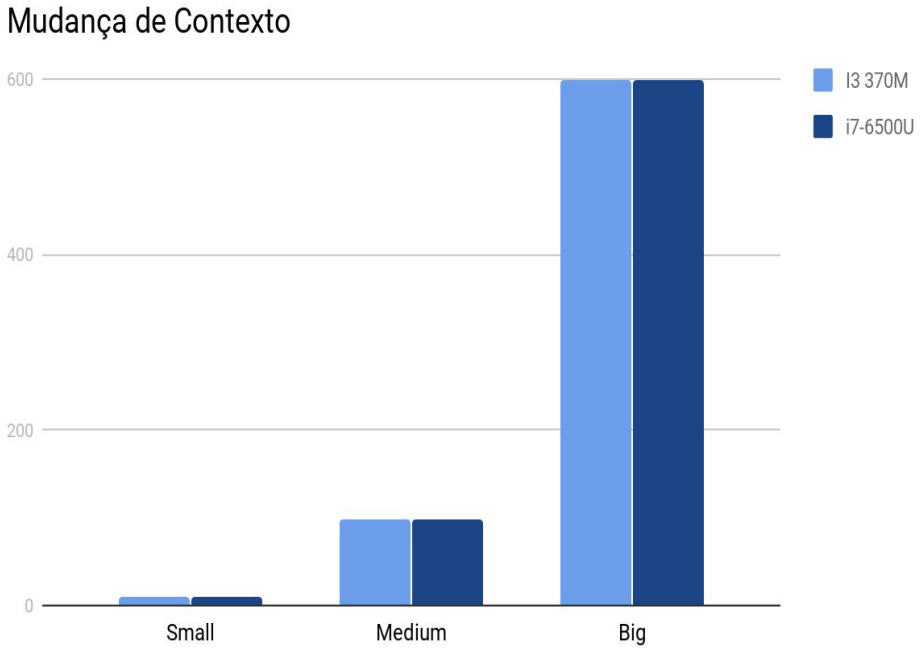
Escalonadores

Para controla as threads usamos conditional variables, para suspender a execução das threads até serem liberadas pelo escalonador.

Para controle utilizamos uma fila que pode ser “convertida” em heap.

First Come First Served



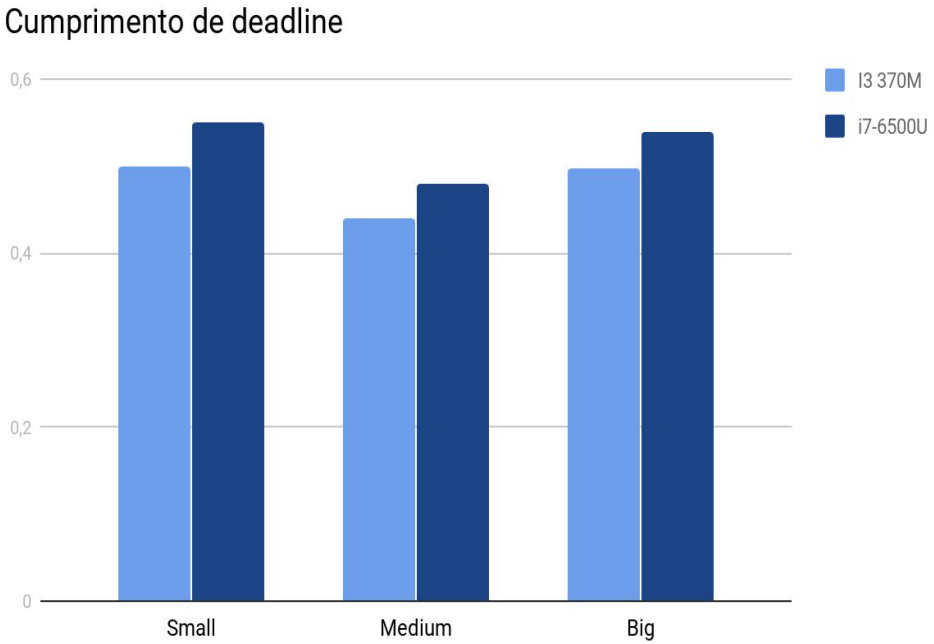


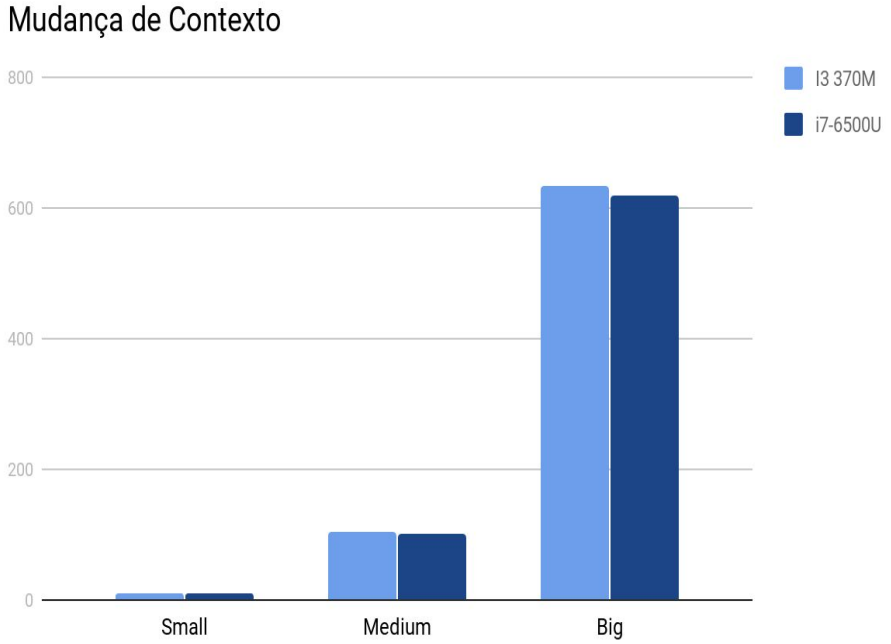
First Come First Served

Intervalo de confiança

I3 370M: Para small.txt, o cumprimento de deadline foi 0.2 ± 0 e mudança de contexto foi 9 ± 0. Para medium.txt, o cumprimento de deadline foi 0.05 ± 0 e mudança de contexto foi 99 ± 0. Para big.txt, o cumprimento de deadline foi 0.0167 ± 0 e mudança de contexto foi 599 ± 0.

Shortest Remaining Time Next



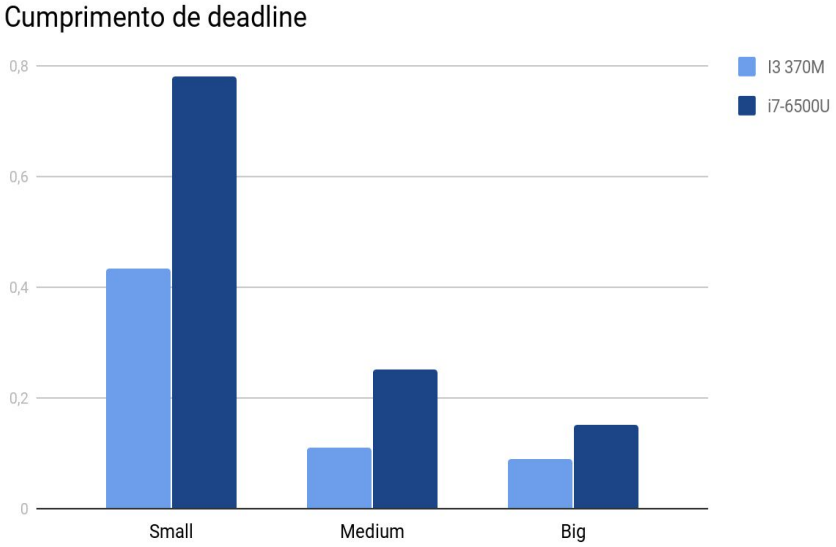


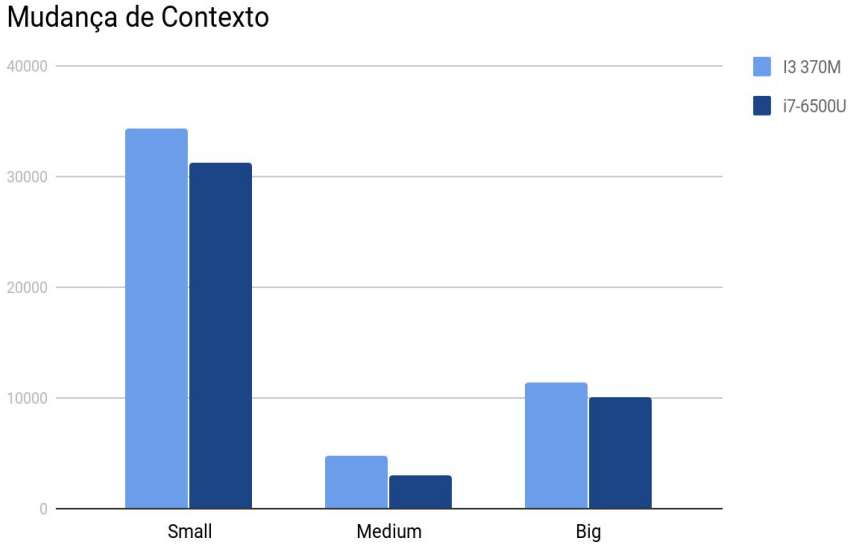
Shortest Remaining Time Next

Intervalo de confiança

I3 370M: Para small.txt, o cumprimento de deadline foi 0.5 ± 0 e mudança de contexto foi 10.03 ± 0.0653. Para medium.txt, o cumprimento de deadline foi 0.44 ± 0 e mudança de contexto foi 105.2 ± 0.18. Para big.txt, o cumprimento de deadline foi 0.498 ± 0 e mudança de contexto foi 632.76 ± 1.22.

Round Robin


Cumprimento de deadline
0,8
0,6
0,4
0,2
0
Small
Medium
Big
I3 370M
i7-6500U


Mudança de Contexto
40000
30000
20000
10000
0
Small
Medium
Big
I3 370M
i7-6500U

Round Robin

I3 370M: Para small.txt, o cumprimento de deadline foi 0.433 ± 0.07 e mudança de contexto foi 34334.3 ± 0.0653. Para medium.txt, o cumprimento de deadline foi 0.109 ± 0.0404 e mudança de contexto foi 4708.63 ± 3071.3. Para big.txt, o cumprimento de deadline foi 0.091 ± 0.0404 e mudança de contexto foi 11383.3

± 5845.073.

Análise dos resultados

Para o First Come First Served, os resultados são constantes. O cumprimento de deadline é constante e ao longo que o número de processos aumenta, o cumprimento de deadline diminui, pois o nosso gerador acabou se tornando ineficiente para calcular deadlines dos processos de arquivos maiores. Também o número de mudança de contexto é constante pois só muda ao terminar o primeiro processo, assim é número de processos menos um.

Para o Shortest Running Time, a quantidade de processos foi proporcional a mudança de contexto, enquanto o cumprimento foi mais constante