EP1 - MAC0422 - 2015
ep1sh e simproc

Renato Lui Geh e Ricardo Fonseca

Shell

No shell ep1sh, além das diversas funções para utilidades, como `extract_args` e `cmd_exists`, as duas funções mais importantes são `run_cmd` e `run_ext_cmd`, que cuidam da execução dos comandos.

run_cmd

`run_cmd` roda os comandos embutidos (`cd`, `pwd` e `exit`), este último adicionado para facilitar os testes durante a produção do EP. Nessa função foi feita manualmente apenas `cd`, já que em `pwd` podíamos apenas chamar `getcwd` e em seguida imprimirmos na saída padrão, e em `exit` podemos chamar apenas `exit(EXIT_SUCCESS)`. Foram implementados os símbolos especiais `..`, `~` e `\` em `cd`.

run_ext_cmd

run_ext_cmd roda os comandos \bin\ls -1 e qualquer executável que esteja em path. Para isso cria-se um novo processo com fork e em seguida usa-se o execve, fazendo o processo pai esperar o processo filho com waitpid.

Funções auxiliares

Para certas tarefas de manipulação de strings e para saber se existe um comando que foi lido foram feitas duas funções: `cmd_exists` e `extract_args`.

cmd_exists

Verifica se o comando que o usuário deseja executar é um comando embutido no shell (cd, pwd ou exit. Se for, roda-se `run_cmd`, senão precisamos rodar `run_ext_cmd`.

extract_args

Separa a linha lida da entrada em tokens separados por espaço. Cada token é então posta em uma posição na tabela de argumentos `args_table`. Considera-se o argumento na posição zero como sendo o próprio comando.

Importante: a linha lida é então inutilizada, já que usa `strtok`. Em `main`, a cada iteração liberamos a memória alocada na variável da linha lida.

history

Por último utilizamos as bibliotecas readline e GNU history, as quais auxiliam para que nosso shell se comporte como o prompt do linux (as setas direcionais auxiliam na edição do texto)

Simulador de processos

Para facilitar o trabalho da implementação do nosso Simulador de Processos, criamos dois arquivos: `utils.c` e `utils.h`. Nesses arquivos foram criados filas, filas de prioridade e uma estrutura para o processo.

Filas foram implementadas num vetor de forma cíclica para melhorar a complexidade. Filas de prioridade foram implementadas como heaps. Chamamos de `thread_clock` o tempo de execução do escalonador.

Escalonadores

Para ser mais fácil de se escolher qual escalonador usar, criamos a variável `manager`, que é um ponteiro para função. Os argumentos de linha de comando são lidos e a função é atribuída a `manager` em seguida.

Além disso alguns escalonadores utilizam filas de prioridade que outros não precisam. Por isso algumas filas nunca são utilizadas se usarmos certo escalonador.

Filas

Em todos os escalonadores implementados foram usadas as filas `p_queue` e `trace_procs`.

`p_queue`: guarda todos os processos que já foram rodados ou que estão rodando ainda.

`trace_procs`: guarda todas que ainda não foram rodadas.

First-Come First-Serve (FCFS)

Este escalonador usa apenas filas. Quando o `thread_clock` chega no instante `t0` de um processo, se há alguma CPU não utilizada, o processo é atribuído àquela CPU.

O processo permanece na CPU até terminar.

Shortest Job First (SJF)

Este escalonador usa uma fila de prioridade com critério de comparação igual a relação de grandeza entre os dts de cada processo concorrente.

Se não há CPU disponível, o processo entra na fila de prioridade e espera ter uma CPU livre.

Shortest Remaining Time Next (SRTN)

Este escalonador é similar ao SJF, com a única diferença que a fila de prioridade tem critério de comparação entre as grandezas dos `thread_clock - deadline` de cada processo.

Se não há CPU disponível, o processo entra na fila de prioridade e espera ter CPU livre.

Round-Robin (RR)

Este escalonador disponibiliza um quantum de tempo para cada processo em cada CPU trabalhando. No `simproc` usamos um quantum `SCHED_QUANTUM=0.050` segundos.

Usou-se `pthread_cond_t` e `pthread_mutex_t` para mandar sinais para suspender ou resumir o thread.