Relatório - Trabalho 2

1. Introdução

O objetivo deste trabalho foi evoluir o simulador de um sistema operacional simples (monotarefa) para um sistema multiprogramado capaz de gerenciar múltiplos processos simultaneamente. O foco principal foi a implementação de processos, a lógica de troca de contexto, o tratamento de chamadas de sistema para criação, gerenciamento e término de processo e a implementação de algoritmos de escalonamento (Round Robin e de Prioridade).

2. Estruturas de Dados e Arquitetura

2.1. Processos

Foi definida a estrutura *processo_t* (em *processo.h*), que armazena todo o contexto de execução de um programa. Os principais campos incluídos foram:

- **Identificação:** *pid* (Process ID) e *parentPID*.
- **Estado do Processador:** Cópia dos registradores (*regA*, *regX*, *regPC*, *regERRO*) para salvar o contexto quando o processo é interrompido.
- **Estado do Processo:** Enumeração indicando se o processo está EXECUTANDO, PRONTO ou BLOQUEADO.
- **Métricas:** Contadores para tempos de execução, espera, bloqueio e número de preempções, úteis para a análise de desempenho.

2.2. Tabela de Processos

O sistema operacional mantém um vetor de ponteiros *processo_t** processosCPU* com tamanho *MAX_PROC*. Este vetor armazena os processos existentes (válido ressaltar que não há diferenciação entre processos mortos e processos que nunca existirão, sem implementação de processo zumbi) . A alocação de PIDs é sequencial.

2.3. Filas de Escalonamento

Para gerenciar a ordem de execução, foram implementadas estruturas de dados dinâmicas:

- **Fila Round Robin:** Uma fila simples (FIFO) para o escalonamento circular.
- **Fila de Prioridades:** Um *Heap* (árvore binária) para ordenar processos baseados em prioridade dinâmica.

3. Núcleo

3.1. Troca de Contexto

A troca de contexto foi implementada nas funções *so_salva_estado_da_cpu* e *so_despacha*:

1. **Salvamento:** Ao ocorrer uma interrupção (IRQ), os registradores da CPU física são lidos da memória (endereços reservados) e salvos na *struct* do *processoCorrente*.
2. **Despacho:** Após o escalonador escolher o próximo processo, os valores armazenados na *struct* desse processo são escritos nos endereços reservados da memória física. A instrução *RETI* da CPU então carrega esses valores para os registradores reais.

3.2. Tratamento de Interrupções

O tratador de interrupções (*so_trata_irq*) foi expandido para lidar com:

- **IRQ_RESET:** Carrega o processo init e configura o timer.
- **IRQ_RELOGIO:** Decrementa o *quantum* do processo atual. Se zerar, ocorre preempção (processo vai para PRONTO e o escalonador é chamado). Também contabiliza métricas de tempo.
- **IRQ_SISTEMA (Syscalls):**
    - SO_CRIA_PROC: Carrega um programa do disco para a memória e cria um novo processo.
    - SO_MATA_PROC: Libera recursos do processo e notifica o pai (se estiver esperando).
    - SO_ESPERA_PROC: Bloqueia o processo pai até que o filho alvo termine.
    - SO_LE / SO_ESCR: Gerencia E/S, bloqueando o processo caso o dispositivo esteja ocupado.

3.3. Tratamento de pendências

A fim de realizar o tratamento de pendências com relação à ocupação dos terminais de E/S, foi criado um indicador dentro do próprio processo para indicar qual terminal ele estava aguardando. As pendências de E/S eram então tratadas da seguinte maneira:

1. Verificação se algum processo está esperando por um terminal;
2. Verificação se o terminal está disponível, e se sim, é feita a leitura ou escrita dos dados;
3. Se mais nenhum processo estiver usando o terminal, o terminal é liberado e o processo desbloqueado;

3.4. Algoritmos de Escalonamento

Dois algoritmos foram implementados e podem ser alternados via definição de compilação:

1. **Round Robin (RR):** Cada processo recebe uma fatia de tempo (*quantum*). Se não terminar nesse tempo, é interrompido e vai para o fim da fila.
2. **Prioridade com Envelhecimento:** Processos possuem uma prioridade base. Para evitar a possibilidade de uma execução semelhante à de um sistema monoprogramado, implementou-se um mecanismo de cálculo dinâmico de prioridade onde processos que esperam muito tempo têm sua prioridade aumentada temporariamente.

4. Resultados e Métricas

O sistema coleta métricas detalhadas ao final da execução de cada processo, incluindo:

- **Tempo de Retorno:** Tempo total desde a criação até o término.

- **Tempo de Resposta:** Tempo até a primeira execução.
- **Tempo de CPU vs. Bloqueio:** Permite analisar se o processo utiliza mais CPU ou E/S.

Os testes realizados com os programas p1 (muita CPU, pouca E/S) e p2/p3 (bastante E/S) demonstraram que o escalonador preemptivo funcionou corretamente, alternando entre processos e mantendo a CPU ocupada enquanto outros processos aguardavam E/S.

5. Conclusão

O desenvolvimento do T2 permitiu a compreensão prática de processos, escalonadores e gerenciamento de dispositivos de E/S. A maior dificuldade encontrada foi o gerenciamento correto dos estados de bloqueio (especialmente em operações de E/S e espera de processos filhos), garantindo que nenhum processo ficasse "perdido" ou fosse escalonado indevidamente, e com relação à morte dos processos, pois era necessário garantir a limpeza do sistema de todos os ponteiros que poderiam ter algum vínculo com o processo morto, como os escalonadores.