Universidade do Vale do Itajaí
Disciplina: Sistemas Operacionais
Avaliação M1 - Sincronização, Processos e Threads

Instruções

1. Esta avaliação deve ser feita em trio e não serão aceitos trabalhos com grupos com quantidades de integrantes diferentes.
2. Data de entrega: 16/09/2025 até as 23:59. Não pode entregar em atraso.
3. Esta avaliação tem por objetivo consolidar o aprendizado sobre conceitos de IPC, processos, threads, concorrência e paralelismo.
4. A implementação deverá ser desenvolvida utilizando a linguagem de programação de sua preferência (C, C++, Python). Porém, a utilização e suporte a IPC, processos e threads pela linguagem escolhida é de responsabilidade do(s) aluno(s), seja usado corretamente o conceito de processos, threads e IPC.
5. As bibliotecas usadas devem ser equivalentes a Pthreads para threads e `fork()` para processos. Bibliotecas que também implementem e que permitam usar conceitos de paralelismo também podem ser usadas, mas o aluno também é responsável pelo seu uso e apresentação.
6. O sistema deve ser entregue em pleno funcionamento. Projetos que não compilem ou que apresentarem falhas de execução receberão nota zero. ” É de responsabilidade do(s) aluno(s) apresentar a execução funcionando corretamente.
7. Deve ser disponibilizado os códigos da implementação em repositório do(s) aluno(s) (github), deve ser fornecido o link e o repositório dever ser público.
8. O relatório deve seguir o formato de artigo científico ou seguindo as normas da ABNT e as orientações para produção de trabalhos acadêmicos da Univali contendo:
    - Identificação do autor e do trabalho.
    - Enunciado do projeto.
    - Explicação e contexto da aplicação para compreensão do problema tratado pela solução.
    - Resultados obtidos com as simulações.
    - Códigos importantes de implementação.
    - Resultados obtidos com a implementação (tabelas, gráficos e etc).
    - Análise e discussão sobre os resultados finais.
9. Deve ser disponibilizado os códigos da implementação juntamente com o relatório (salvo o caso da disponibilidade em repositório aberto do aluno, que deve ser fornecido o link). O repositório deve estar aberto do momento da entrega em diante, sendo que o professor não se responsabiliza caso o projeto não esteja disponível para consulta no momento da correção, sendo do(s) aluno(s) essa responsabilidade de manter disponível. Cópias entre alunos implicará em nota zero para todos os envolvidos.
10. O trabalho deverá ser apresentado em data definida pelo professor. É de responsabilidade do(s) aluno(s) explicar os conceitos, comandos, bibliotecas usadas. É de responsabilidade do(s) aluno(s) fazer a solução funcionar e ela deverá ser baixada do local de entrega no momento da apresentação. Os trabalhos não apresentados terão como nota máxima 5,0, além dos descontos aplicados no restante da avaliação da implementação. Todos os alunos poderão apresentar ou o professor poderá escolher um representante para apresentar o trabalho em nome do grupo.

Descrição do projeto a ser desenvolvido

Problemática: Sistema de Gerenciamento de Fila de Impressão Paralelo

Descrição Geral

Você irá desenvolver um sistema que simula um servidor de impressão (spooler), que gerencia múltiplos trabalhos de impressão enviados por diferentes processos clientes. O sistema deverá usar processos para os clientes, um processo para o servidor e um conjunto de threads para simular as impressoras que processam os trabalhos em paralelo.

- **Múltiplos processos clientes** enviam solicitações de impressão para um processo servidor central via IPC (pipe ou memória compartilhada).
- O **processo servidor** recebe as solicitações e as armazena em uma fila de impressão compartilhada.
- O servidor gerencia um **pool de threads**, onde cada thread representa uma impressora. Essas threads consomem os trabalhos da fila e os "imprimem" (simulam o processamento).
- As threads utilizam **mutex ou semáforo** para garantir acesso seguro à fila de impressão, evitando que múltiplos trabalhos sejam pegos ao mesmo tempo ou que a fila seja corrompida.

Componentes do Sistema

1. **Processo Cliente**
    - Cada cliente é um processo separado que gera e envia "trabalhos de impressão" para o servidor.
    - Exemplo de trabalho de impressão: um `struct` contendo `id_job`, `nome_arquivo` e `numero_paginas`.
    - A comunicação com o servidor é feita via pipe ou memória compartilhada.
2. **Processo Servidor (Gerenciador de Impressão)**
    - Lê os trabalhos de impressão enviados pelos clientes via IPC.
    - Adiciona os trabalhos recebidos a uma fila compartilhada (ex: lista ligada, vetor dinâmico).
    - Cria e gerencia um pool de threads (impressoras) que processam os trabalhos da fila.
    - Registra a atividade (recebimento e conclusão de trabalhos) em um arquivo de log.

Estrutura do Trabalho de Impressão

C
```
typedef struct {
    int id_job;
    char nome_arquivo[50];
    int numero_paginas;
} TrabalhoImpressao;
```

Uso de Threads e Mutex/Semáforo

- As threads (impressoras) compartilham a mesma fila de trabalhos de impressão.
- O acesso à fila (para adicionar ou remover trabalhos) é uma região crítica e deve ser protegido por:
    - `pthread_mutex_t` (para garantir que apenas uma thread/processo modifique a fila por vez), ou
    - `sem_t` (para controlar o número de trabalhos na fila e coordenar as threads produtoras/consumidoras).
- Isso evita condições de corrida e garante a consistência da fila de impressão.

Fluxo do Sistema

```
[Processo Cliente cria e envia Job]
                ↓
[Pipe ou Memória Compartilhada]
                ↓
[Processo Servidor recebe o Job e enfileira]
                ↓
[Fila de Impressão com Mutex/Semáforo]
                ↓
[Threads (Impressoras) consomem e processam os Jobs]
                ↓
[Arquivo de Log com o status dos trabalhos]
```

Estrutura de Arquivos Sugerida

```
spooler_impressao/
├── cliente.c          # Gera e envia trabalhos de impressão
├── servidor.c         # Recebe trabalhos, gerencia a fila e as threads
├── fila.h             # Definição da estrutura da fila e do job
├── log_servidor.txt   # Arquivo de log gerado pelo servidor
```

Conceitos Praticados

- Comunicação entre Processos (IPC) com pipe ou memória compartilhada.
- Criação e gerenciamento de múltiplos processos (`fork()`).
- Criação e gerenciamento de threads para processamento paralelo.
- Sincronização e controle de concorrência com mutex ou semáforos.
- Simulação de um sistema produtor-consumidor (clientes produzem, threads consomem).
- Gerenciamento de recursos compartilhados em um ambiente concorrente.