Trabalho I – Criação, Monitoramento e Comunicação de Processos e /Threads

Trabalho I – Regras

Grupos

- o trabalho pode ser realizado em grupos de até 3 alunos

O que entregar

- código fonte (comentado) na linguagem de preferência
- instruções de compilação (se necessário) e execução do código

Quando entregar e Onde entregar

- tarefa no AVA até 20/05/2025 23h55

Peso

- 50% da Nota 1 (1º bimestre)
  - a nota do trabalho é composta de duas partes
    1. trabalho em si (grupo)
    2. apresentação (individual)

Trabalho I – Regras

Sistema

- o Sistema Operacional alvo para o qual a aplicação será escrita é de livre escolha
  - dicas de bibliotecas
    - Linux: `sys/resource.h` , `unistd.h`, `sys/sysinfo.h`, `pthreads.h`
    - Windows: `Sysinfoapi.h`, `processthreadsapi.h`

Apresentação/Demonstração em aula (data a definir)

- os alunos devem ser capazes de demonstrar o funcionamento do seu programa em sala de aula
- todos os alunos devem estar aptos a apresentar qualquer parte do trabalho

Atenção

- todas as entregas serão submetidas a verificação de plágio por “medida de similaridade de código”

Trabalho I – Requisitos

- O trabalho terá **requisitos** mínimos e recomendados
  - trabalhos que implementem apenas os requisitos mínimos terão nota oscilando próximo à média (7)
  - para alcançar notas maiores será necessário implementar (parte) dos requisitos recomendados
  - para trabalhos desenvolvidos em grupo espera-se mais funcionalidades que aqueles feitos de forma individual

Trabalho I – Especificação

- Implemente um programa (de nome FMS) capaz de lançar e controlar a execução de um outro programa conforme os requisitos descritos

Trabalho I – Especificação

- Implemente um programa (de nome FMS) capaz de lançar e controlar a execução de um outro programa conforme os requisitos descritos
- **requisitos mínimos**
  1. lançar a execução de qualquer programa executável cujo binário seja fornecido pelo usuário
     - deve existir uma interface (textual ou gráfica) para que o nome do binário seja informado pelo(a) usuário(a) sem que seja necessário editar o código fonte

Trabalho I – Especificação

- Implemente um programa (de nome FMS) capaz de lançar e controlar a execução de um outro programa conforme os requisitos descritos
- **requisitos mínimos**
  1. lançar a execução de qualquer programa executável cujo binário seja fornecido pelo usuário
     - deve existir uma interface (textual ou gráfica) para que o nome do binário seja informado pelo(a) usuário(a) sem que seja necessário editar o código fonte
  2. consultar a(o) usuária(o) sobre
     - a quota de tempo de CPU para execução do binário fornecido
     - eventual tempo limite (timeout) para execução do binário fornecido
     - um eventual montante máximo de memória para execução do binário fornecido

Trabalho I – Especificação

- Implemente um programa (de nome FMS) capaz de lançar e controlar a execução de um outro programa conforme os requisitos descritos
- **requisitos mínimos**
  1. lançar a execução de qualquer programa executável cujo binário seja fornecido pelo usuário
     - deve existir uma interface (textual ou gráfica) para que o nome do binário seja informado pelo(a) usuário(a) sem que seja necessário editar o código fonte
  2. consultar a(o) usuária(o) sobre
     - a quota de tempo de CPU para execução do binário fornecido
     - eventual tempo limite (timeout) para execução do binário fornecido
     - um eventual montante máximo de memória para execução do binário fornecido
  3. monitorar a execução do processo criado para executar o binário fornecido
     - identificar quando o programa terminou
     - quantificar o tempo de CPU (usuário e sistema) utilizado
     - quantificar o máximo de memória utilizado
     - controlar o tempo de relógio e matar o processo caso o *timeout* expire

- **requisitos mínimos**
  4. o programa FMS deve funcionar em laço, solicitando um novo binário sempre que ainda houver quota de tempo de CPU disponível
     - caso a quota de CPU ou o consumo máximo de memória seja ultrapassada(o), o programa deve reportar tal situação
     - o FMS deve encerrar caso algum dos limites (CPU ou memória) seja extrapolado
     - porém a expiração do *timeout* não encerra o FMS, apenas o programa monitorado

- **requisitos mínimos**
  4. o programa FMS deve funcionar em laço, solicitando um novo binário sempre que ainda houver quota de tempo de CPU disponível
     - caso a quota de CPU ou o consumo máximo de memória seja ultrapassada(o), o programa deve reportar tal situação
     - o FMS deve encerrar caso algum dos limites (CPU ou memória) seja extrapolado
     - porém a expiração do *timeout* não encerra o FMS, apenas o programa monitorado
- **requisitos recomendados**
  1. utilizar diferentes *threads* para monitorar constantemente (e.g., 1x por segundo) o consumo de quota de CPU e o uso máximo de memória
     - matando o programa caso ao menos um dos limites seja extrapolado

Trabalho I – Especificação

- **requisitos mínimos**
  4. o programa FMS deve funcionar em laço, solicitando um novo binário sempre que ainda houver quota de tempo de CPU disponível
     - caso a quota de CPU ou o consumo máximo de memória seja ultrapassada(o), o programa deve reportar tal situação
     - o FMS deve encerrar caso algum dos limites (CPU ou memória) seja extrapolado
     - porém a expiração do *timeout* não encerra o FMS, apenas o programa monitorado
- **requisitos recomendados**
  1. utilizar diferentes *threads* para monitorar constantemente (e.g., 1x por segundo) o consumo de quota de CPU e o uso máximo de memória
     - matando o programa caso ao menos um dos limites seja extrapolado
  2. reportar ao usuário o progresso do consumo de quota de CPU e de memória

Trabalho I – Especificação

- **requisitos mínimos**
  4. o programa FMS deve funcionar em laço, solicitando um novo binário sempre que ainda houver quota de tempo de CPU disponível
     - caso a quota de CPU ou o consumo máximo de memória seja ultrapassada(o), o programa deve reportar tal situação
     - o FMS deve encerrar caso algum dos limites (CPU ou memória) seja extrapolado
     - porém a expiração do *timeout* não encerra o FMS, apenas o programa monitorado
- **requisitos recomendados**
  1. utilizar diferentes *threads* para monitorar constantemente (e.g., 1x por segundo) o consumo de quota de CPU e o uso máximo de memória
     - matando o programa caso ao menos um dos limites seja extrapolado
  2. reportar ao usuário o progresso do consumo de quota de CPU e de memória
  3. não descontar da quota, execuções que falhem em lançar o binário fornecido

Trabalho I – Especificação

- **requisitos mínimos**
  4. o programa FMS deve funcionar em laço, solicitando um novo binário sempre que ainda houver quota de tempo de CPU disponível
     - caso a quota de CPU ou o consumo máximo de memória seja ultrapassada(o), o programa deve reportar tal situação
     - o FMS deve encerrar caso algum dos limites (CPU ou memória) seja extrapolado
     - porém a expiração do *timeout* não encerra o FMS, apenas o programa monitorado
- **requisitos recomendados**
  1. utilizar diferentes *threads* para monitorar constantemente (e.g., 1x por segundo) o consumo de quota de CPU e o uso máximo de memória
     - matando o programa caso ao menos um dos limites seja extrapolado
  2. reportar ao usuário o progresso do consumo de quota de CPU e de memória
  3. não descontar da quota, execuções que falhem em lançar o binário fornecido
  4. monitorar a árvore de processos criados
     - útil para programas que criam muitos processos

- **requisitos recomendados**

5. implementar ”operação pré-paga“ com base em créditos

Fonte: openclipart.org

- **requisitos recomendados**

5. implementar ”operação pré-paga“ com base em créditos

Fonte: openclipart.org

6. implementar ”operação pós-paga“ pague pelo uso

Fonte: openclipart.org

- atenção:
  - o tempo de CPU é diferente do tempo de relógio (*walltime*)
    - o tempo de relógio é o tempo real transcorrido entre o início e o fim da execução
    - o tempo de CPU é o tempo em que a CPU ficou ocupada de fato executando código do processo diretamente ou por intermédio do sistema
  - a função *timeout* não necessariamente evita que a quota de CPU seja superada

Demonstração

- assista a demonstração!