COPPE UFRJ

GTA / UFRJ

Comunicação entre processos

Pedro Cruz

EEL770 – Sistemas Operacionais

Aviso

GTA / UFRJ

- Os problemas e as soluções que veremos nessa aula servem para:
  - Processos
  - *Threads*
  - Agentes em um sistema distribuído

Interação entre processos

GTA / UFRJ

- Comunicação entre processos (*interprocess communication* – IPC)
  - Troca de dados
  - Consistência
  - Sincronização

Interação em *threads* no mesmo processo

- Troca de dados
  - Trivial
    - Área de memória compartilhada serve de canal
- Consistência
  - Que nem em processos
- Sincronização
  - Que nem em processos

Comunicação trivial

GTA / UFRJ

- Operacional fornece área compartilhada de memória
  - Processos escrevem e leem a área
  - Processos podem sobrescrever mensagens de outros
    - Acidentalmente

- Exemplo:
  - Processos escrevem em memória arquivos para impressão
    - Fila
  - Processo de impressão verifica a memória periodicamente
  - Escalonador alterna a execução dos processos

Exclusão mútua e sincronização

GTA / UFRJ

- Exclusão mútua
  - Queremos garantir que um recurso seja acessado apenas por um processo de cada vez
- Sincronização
  - Queremos garantir que processos sigam uma determinada ordem de processamento

Condição de corrida

- "Condição de concorrência"
  - Saída dos processos depende de uma sequência de eventos
    - Pode haver erros

- Depurar programas assim é muito difícil

Chaveamento entre processos

- Escalonador
  - Decide qual processo vai ser executado no processador

- É ativado por um relógio
  - Relógio gera interrupção
  - Operacional é chamado para tratar interrupção
  - Operacional põe escalonador no controle
  - Escalonador chaveia entre processos

Regiões críticas

GTA / UFRJ

- Sequência de instruções de um mesmo processo
  - Memória compartilhada é acessada
  - Memória compartilhada está inconsistente

Regiões críticas - exemplo

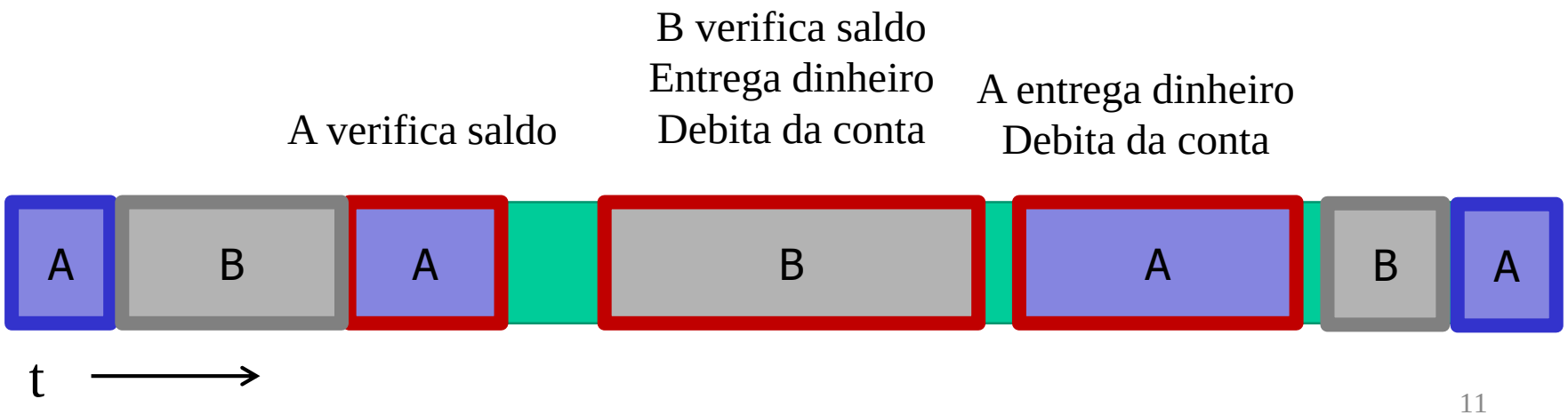
GTA / UFRJ

- Cliente realiza saque no caixa eletrônico
  - Verificação de saldo
  - Entrega do dinheiro
  - Débito da conta corrente

Regiões críticas

GTA / UFRJ

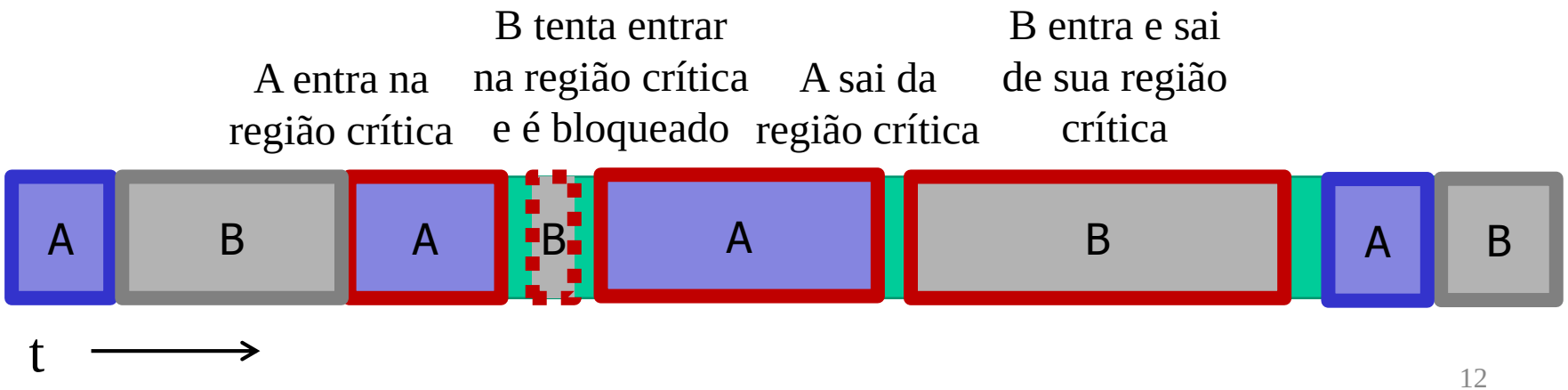
- Caixa A retira dinheiro
  - Verifica saldo
  - Entrega dinheiro
  - Debita da conta corrente
- Caixa B retira dinheiro
  - Verifica saldo
  - Entrega dinheiro
  - Debita da conta corrente



Exclusão mútua

GTA / UFRJ

- Garantia de acesso único a regiões compartilhadas
  - Dois processos nunca estão em regiões críticas simultaneamente
  - Apenas regiões críticas causam bloqueio de outros processos
  - Um processo sempre sai de sua região crítica
  - Quantidade e desempenho de CPU's não deve importar



Garantia de exclusão mútua

GTA / UFRJ

- Espera ocupada

- Dormir e acordar

Espera ocupada

GTA / UFRJ

- Um processo entra em sua região crítica
  - Nenhum outro processo entra em região crítica até que o primeiro saia
  - Outro processo que precise entrar em região crítica realiza uma espera ocupada
    - Suspende todas as suas operações para testar variável que indica se pode ou não entrar na região crítica

```
while(!allowed){}
critical_op();  //Realiza operação na região crítica
```

Qual o problema?

Opções com espera ocupada

GTA / UFRJ

- Desabilitar interrupções
- Variáveis do tipo trava
- Variáveis do tipo "vez"
- Solução de Peterson
- Instrução Testar e Configurar Trava
  - *Test and Set Lock* – TSL
- Instrução XCHG

Desabilitar interrupções

GTA / UFRJ

- Processo em região crítica não é interrompido
  - Nem pelo operacional (!!!)
  - Escalonador não é acordado

- Vantagem
  - Garante que não vai haver interrupções
    - Nem do escalonador
- Desvantagem
  - Processo mal-intencionado pode se aproveitar

- O operacional pode usar isso em operações internas

Variáveis do tipo “trava”

GTA / UFRJ

- Variável compartilhada
  - Processo quer entrar em região crítica -> testa variável
    - Se 1, existe outro processo em região crítica
      - Processo entra em espera ocupada
    - Se 0, não existe outro processo em região crítica
      - Processo configura trava para 1
      - Processo executa a região crítica
      - Processo configura a trava para 0

Qual o problema?

Problema da operação não-atômica

- Ler -> escrever não é uma operação atômica
  - Se processo realiza uma leitura e é interrompido antes de fazer uma escrita, temos um problema

Variáveis do tipo "vez"

GTA / UFRJ

- Variável "vez" é compartilhada entre processos
  - Cada processo passa a vez após terminar sua região crítica

```
#define N_PROC 0
#define NEXT 1
while(true){
        while(turn!=N_PROC){}
        critical_op();
        turn = NEXT;
        non_critical_op();
}
```

Qual o problema?

Variável do tipo “vez” – problema

- Processos devem sempre entrar em suas regiões críticas de forma alternada
  - Processo fica bloqueado até que todos os outros entrem e saiam de suas regiões críticas

Solução de Peterson

- Duas funções são chamadas por processos
  - Uma para pedir para entrar na região crítica
    - Aguarda se não for possível entrar na região crítica
  - Outra para liberar os outros processos

Solução de Peterson para dois processos


```
int turn;
bool interested[2];
void enter_region(int p_id, other_id){
  interested[p_id] = True;
  turn = p_id;
  while(turn == p_id &&
       interested[other_id]==True){}
}
void leave_region(int p_id){
  interested[p_id] = False;
}
```

Solução de Peterson para mais processos

- Processos se registram em uma "fila"
- Processos que estão na fila ficam bloqueados
- Processos que não estão na fila não ficam bloqueados

Instrução TSL

GTA / UFRJ

- Verifica trava e configura trava em operação atômica
- Instrução bloqueia acesso de outros processadores ou núcleos ao barramento
  - Ambiente com múltiplos processadores

Instrução TSL – uso

GTA / UFRJ

```
enter_region:
   MOV REGISTER, #1          | coloca 1 no registrador
   TSL REGISTER,LOCK         | copia lock para register e
                             | coloca lock em 1;
   CMP REGISTER,#0           | testa se lock valia 0;
   JNE enter_region          | repete, se lock não valia 0;
   RET                       | retorna, com autorização;

leave_region:
   MOVE LOCK,#0              | configura lock para 0;
   RET                       | retorna;
```

Instruções TSL e XCHG

GTA / UFRJ

- Seria possível obter o mesmo efeito utilizando apenas a instrução XCHG?
  - Se sim, como?
  - Se não, qual a diferença?
  - Se "mais ou menos", o que é possível simular e o que não?

Dormir e acordar

GTA / UFRJ

- Espera ocupada é ruim
  - Se processo com prioridade baixa entra em região crítica
    - Bloqueiam outros
    - Demora a ser chamado
      - Demora a sair da região crítica
- Seria melhor se processos pudessem cooperar
  - Ceder tempo de CPU para outro processo liberar logo recursos
    - Implementado na forma de chamadas de sistema
      - **Sleep**
      - **Wakeup**

Dormir e acordar

GTA / UFRJ

- *Sleep*
  - Bloqueia o processo e aguarda evento
    - Evento é uma chamada de *wake up*

- *Wake up*
  - Desbloqueia um processo que realizou chamada *sleep*
    - Se processo não estava dormindo, chamada é perdida

Produtores e consumidores

GTA / UFRJ

- Processo produtor escreve num buffer de tamanho finito
  - Se buffer está cheio -> *sleep*
  - Se não -> produz
    - Insere item no buffer
    - Incrementa variável de tamanho do buffer
    - Se buffer estava vazio -> acorda consumidor (*wakeup)*
- Processo consumidor apaga os dados do buffer
  - Se buffer está vazio -> *sleep*
  - Se não -> consome
    - Remove item do buffer
    - Decrementa variável de tamanho do buffer
    - Se buffer estava cheio -> acorda produtor (*wakeup*)

Produtores e consumidores

GTA / UFRJ

- Condição de corrida possível:
  - Consumidor verifica buffer vazio (vai chamar *sleep*)
  - **ESCALONADOR!**
  - Produtor produz primeiro item do buffer
    - Emite *wakeup*
  - **ESCALONADOR!**
  - Consumidor realiza chamada *sleep*
  - **ESCALONADOR!**
  - Produtor produz até encher o buffer
    - *Sleep*
  - Ambos os processos estão dormindo
    - Um aguardando sinal do outro

Semáforos

GTA / UFRJ

- Variável "compartilhada" diz quantos recursos estão disponíveis
  - Operação de verificar variável, alterar variável e dormir são atômicas
    - Implementadas como chamadas de sistema
      - *Up*
      - *Down*

Up e *Down*

GTA / UFRJ

- *Up*
  - Verifica variável
    - Se possível, incrementa e segue para execução
      - Envia sinal de *wake*
    - Se não, dorme e aguarda sinal
- Down
  - Verifica variável
    - Se possível, decrementa e segue para execução
      - Envia sinal de *wake*
    - Se não, dorme e aguarda sinal

Ocorrem de forma atômica!

Consumidor e produtor com semáforo

GTA / UFRJ

- Produtor executa *up*
  - Se puder, produz
  - Se não, dorme e aguarda *wakeup*
- Consumidor executa *down*
  - Se puder, consome
  - Se não, dorme e aguarda *wakeup*

COPPE UFRJ

GTA / UFRJ

Comunicação entre processos

Pedro Cruz

EEL770 – Sistemas Operacionais