COPPE UFRJ

Threads

Pedro Cruz

EEL770 – Sistemas Operacionais

Nas aulas anteriores

GTA / UFRJ

- "A" *thread* ou "o" *thread*?
  - Não sei. Já vi todos

Chamadas de sistema

GTA / UFRJ

- Pedido de um processo ao Sistema Operacional (SO)
  - Bloqueante
    - Deixa o processo em estado bloqueado
      - Abrir, criar, ler, escrever em arquivos
      - Aceitar conexões
      - Enviar, receber mensagens de outros processos
  - Não bloqueante
    - Não deixa o processo em estado bloqueado
      - Verificar estado de variáveis de ambiente
      - Realizar entrada/saída assíncrona

Processo

GTA / UFRJ

- Gerenciamento de recursos
  - Memória
  - I/O
- Gerenciamento de execução
  - Pilha
  - *PSW* (*program status word*)

Motivação

GTA / UFRJ

- Processo pode ficar bloqueado
  - Operação de I/O
    - Dispositivo
    - Usuário
    - Computador remoto
- Processos não compartilham espaços de endereçamento
- Processo único não pode se aproveitar de múltiplas CPUs

Exemplo

GTA / UFRJ

- Navegador
  - Aguarda usuário digitar
    - Não consegue atualizar tela ou trocar aba
    - Não consegue receber dados da rede
  - Aguarda dados da rede
    - Não consegue receber dados do usuário
    - Não consegue atualizar tela ou trocar aba

Ideia

GTA / UFRJ

- Processos "menores" dentro de um processo
  - Processos menores aguardam desbloqueios
    - Processo maior não é bloqueado
    - Redução do tempo de ociosidade da CPU
    - Melhorias na interatividade com usuário
  - Dados são compartilhados
  - Recursos são compartilhados

Processos vs *Threads*

GTA / UFRJ

- Processos
  - Recursos
  - Execução
- *Threads* – fio/linha/carretel
  - Execução
    - "Processos leves"

Thread

GTA / UFRJ

- Vinculada a um processo
  - Compartilha recursos com o processo
- Possui sua própria pilha de execução
  - Desse ponto de vista, é um processo em separado
- Pode coexistir com outras *threads* do mesmo processo
- Compartilha dados globais com membros do processo

Thread

GTA / UFRJ

- Estado de execução
  - Executando, Pronta, Bloqueada...
- Contexto
- Pilha de execução
- Acesso aos recursos do processo
  - Compartilhado com as outras *threads* do mesmo processo

Criação de uma *thread*

GTA / UFRJ

- Criação das variáveis necessárias, no espaço do processo que a criou
  - Pilha de execução
    - Código que será executado pela *thread*
  - Variáveis locais

Processos vs *Threads* – visão do desenvolvedor

- Seu processo compete com processos de outros desenvolvedores
  - Competição pode ser destrutiva
    - Processos não são necessariamente bem comportados
    - Operacional é o árbitro

- Sua *thread* compete com as **SUAS** *threads*
  - Competição saudável
    - *Threads* são bem comportadas
    - O desenvolvedor é o árbitro

Portable Operating System Interface – POSIX

- Família de padrões da IEEE para sistemas operacionais
  - IEEE 1003.1.c define *threads*

Threads POSIX

GTA / UFRJ

- Pthread_create
  - Cria uma *thread*
- Pthread_exit
  - Conclui uma *thread*
- Pthread_join
  - Espera que uma *thread* termine
- Pthread_yield
  - Libera a CPU para executar outra *thread*
- Pthread_attr_init
  - Inicializa atributos do *thread*
- Pthread_attr_destroy
  - Destrói atributos da *thread*

Atributos de *threads*

GTA / UFRJ

- Normalmente os *default* são suficientes
- Só podem ser modificados na inicialização da *thread*
- Exemplos
  - Tamanho da pilha
  - Endereço da pilha
  - Preservação da *thread* depois de seu término
  - Prioridade

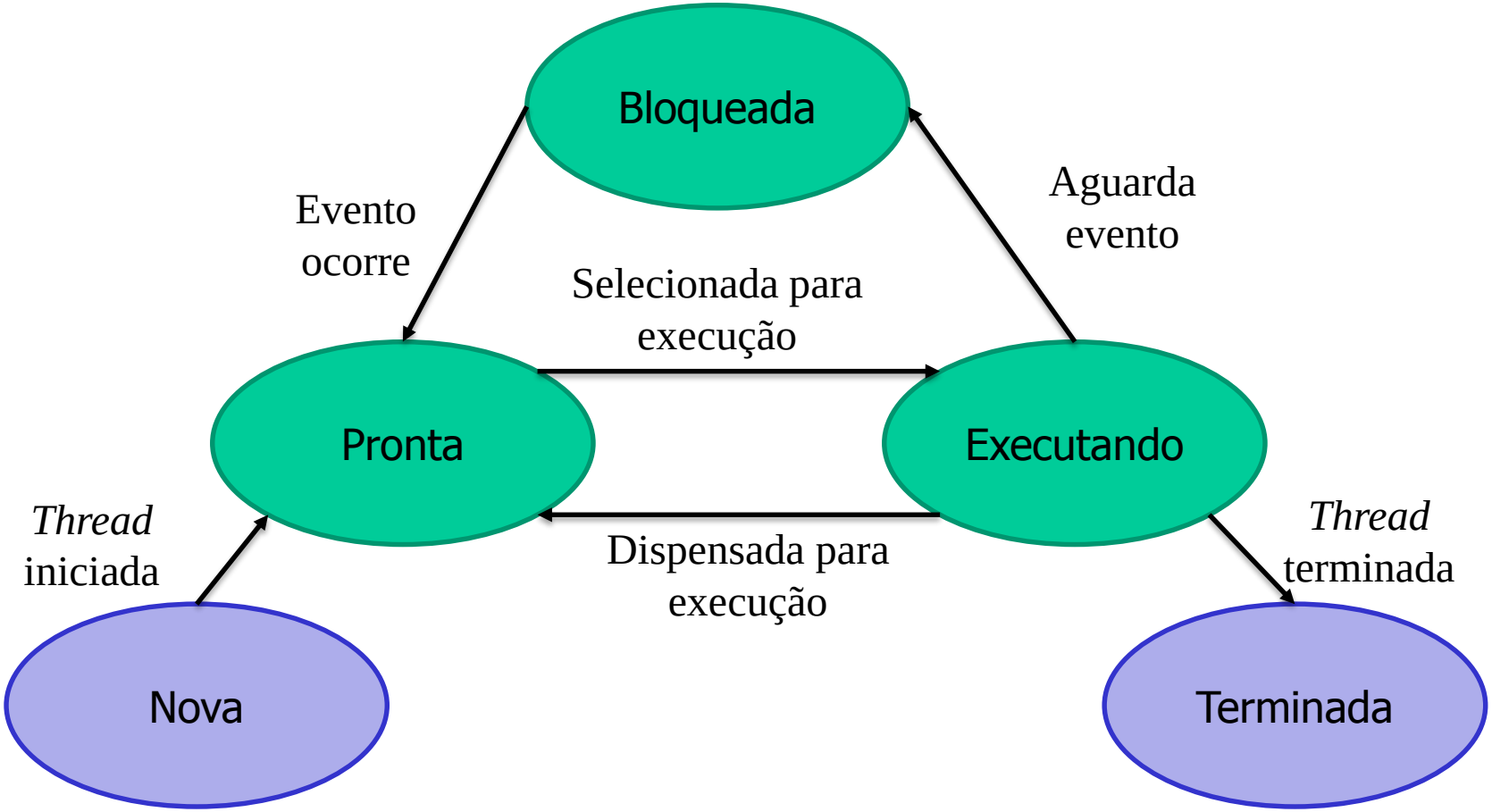
Estados de uma *thread*

GTA / UFRJ

- Nova
  - Acabou de ser criada
- Pronta
  - Está aguardando ser executada
- Executando
  - Está sendo executada pela CPU
- Bloqueada
  - Está aguardando alguma operação de I/O
- Terminada
  - *Thread* já foi executada

Estados de *threads*

GTA / UFRJ



Threads - implementação

GTA / UFRJ

- Espaço do usuário
- Espaço do núcleo
- Implementação híbrida

Espaço usuário

GTA / UFRJ

- Operacional ignora as *threads*
  - Compatibilidade
- Biblioteca implementa gerenciamento de *threads*
  - Sistema de tempo de execução
- Processos devem ter uma tabela de *threads*
- Bloqueios locais de *threads* são avisadas ao sistema de tempo de execução
  - Sistema de tempo de execução chaveia as *threads*

Vantagens do espaço de usuário

- Chaveamento de *threads* é local
  - Não precisa de chaveamento para modo núcleo
    - Muito mais rápido
- Algoritmo de chaveamento é local
  - Pode ser customizado para necessidade local

Problemas do espaço de usuário

- Chamadas bloqueantes não podem ser tratadas
  - Algumas opções pouco elegantes
    - *Select*
      - Chamada UNIX que verifica se buffers de I/O estão cheios
        - » Se estiverem, chamada não bloqueará
  - Chamadas bloqueantes ocorrem "sem motivo"

- Cada *thread* deve ser bem educada
  - Não existe, entre as *threads*, um escalonador onipotente
  - *Thread* não pode ser impedida de utilizar todos os recursos

Threads no espaço de núcleo

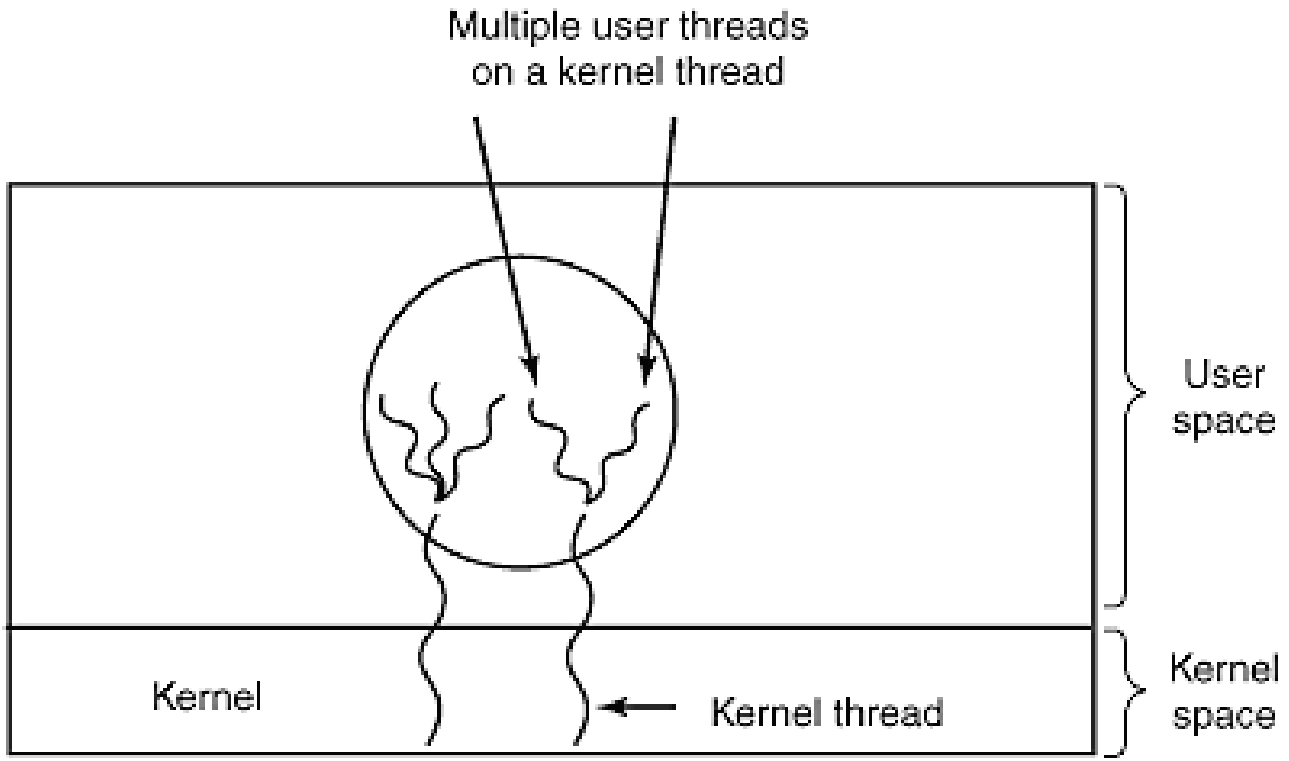
GTA / UFRJ

- Gerenciamento único de *threads*
  - Criação de *threads* é mais custoso
- Chaveamento de *threads*
  - Onisciente
  - Onipotente
  - Onipresente
  - Em modo núcleo
- Chamadas bloqueantes podem ser tratadas

Threads híbridas

GTA / UFRJ

- Várias *threads* no núcleo que podem ser bifurcadas em *threads* de usuário



Fonte: Tanenbaum

Ativação pelo escalonador

GTA / UFRJ

- *Threads* são implementadas no espaço de usuário
- Chamadas de sistema bloqueantes são notificadas ao processo (ao sistema de tempo de execução)
  - *Upcall*
- Sistema de tempo de execução realiza chaveamento de *threads*

Threads pop-up

GTA / UFRJ

- *Threads* executados no espaço de núcleo para tratar eventos
  - Criadas no momento em que o evento é disparado
    - Rápidas de criar
  - Executadas no espaço do núcleo
    - Não há chaveamento

Armadilhas de códigos *multithread*

GTA / UFRJ

- Variáveis globais
- Consistência
- Bibliotecas que não podem ser acessadas "simultaneamente"
- Sinais e outros eventos "por processo"
  - Qual *thread* deveria receber um sinal recebido por um processo?
- Pilha de execução
  - Operacional gerencia a pilha de um processo
  - Processo deve gerenciar as pilhas de *threads* implementadas no espaço de usuário

Threads – visão do desenvolvedor

- Código obrigatoriamente sequencial fica na mesma *thread*

- Código paralelizável pode ficar em *threads* diferentes
  - Compromisso
    - Grau de paralelização
    - Sobrecarga
      - Criação
      - Memória
      - Chaveamento entre *threads*
      - Gerenciamento de recursos

COPPE UFRJ

Threads

Pedro Cruz

EEL770 – Sistemas Operacionais