Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0036 – Sistemas Operacionais

Threads: Implementação

Prof. Gustavo Girão
girao@imd.ufrn.br

Roteiro

- Criação de Threads
- Espaço de endereçamento
- Terminação de Threads
- Thread Join
- Exercícios

Criação de threads no UNIX

- A implementação depende da biblioteca utilizada
  - POSIX -> Pthreads (POSIX Threads)
- **int** pthread_create(**pthread_t *** *thread*, **const pthread_attr_t *** *attr*, **void *** *(*start_routine)(void*)*, **void ***arg*);*
- Argumentos:
  - **thread** – Thread ID.
  - **attr** – NULL para o default.
  - **start_routine**: é uma função que será executada pela thread. Pega em argumento um 'void*' e retorna um void*.
  - **arg** – ponteiro sobre o argumento da função. Para passar mais de um argumento, usar um vetor ou uma struct.

Terminação de Threads

- **int** pthread_exit(**void** ***** *ret*);
- Argumento:
  - **ret** – valor de retorno ao terminar a thread
    - ✧ conceitualmente, este **não** é o retorno da função
- Threads podem terminar:
  - Por ter finalizado sua rotina principal
  - Realizando uma chamada a pthread_exit()
  - Sendo cancelada por outra thread (pthread_cancel)
  - O processo do qual ela faz parte for terminado
    - ✧ Por exit()
    - ✧ Por ter terminado o main()

Espaço de endereçamento

- Quando uma thread é criada, existem dois espaços de endereçamento
  - Compartilhado entre todas as threads
  - Privado e acessível individualmente e exclusivamente thread
- Como identificar estes espaços?
  - Compartilhado
    - ✧ Variáveis globais que são parte da memória de dados
  - Privado
    - ✧ Parâmetros passados na execução da thread
      - Pilha
    - ✧ Variáveis de escopo local

Junção de Threads

- **int** pthread_join(**pthread** *tid*, **void** ** *ret*);
- Argumento:
  - **tid** – ID da thread ao qual se espera
  - **ret** – valor de retorno da thread

- Utilizada para esperar a finalização de uma thread específica
- O valor de retorno recebido da execução da thread é escrito no parâmetro **ret.**
  - O retorno da função seja por **return** ou por **pthread_exit** é escrito em **ret**

Exercício

- Utilizando variáveis compartilhadas como mecanismo de comunicação entre threads, escreva um programa que realiza a soma dos N primeiros números naturais onde:
  - Cada thread realiza exatamente uma das somas
  - A primeira thread imprime o resultado final

Criação de processos com multiplas threads

- O que acontece quando um processo que contem multiplas threads utiliza uma chamada fork?
  - Um novo processo é criado:
    - Cópia da memória de dados
    - Cópia da memória de Instruções
  - Se as threads já foram criadas?
    - São duplicadas
  - Se não foram criadas
    - pthread_join retorna um erro
  - E se uma thread executa um fork??
- As threads criadas são associadas ao processo que as criou.

Escalonamento de Threads

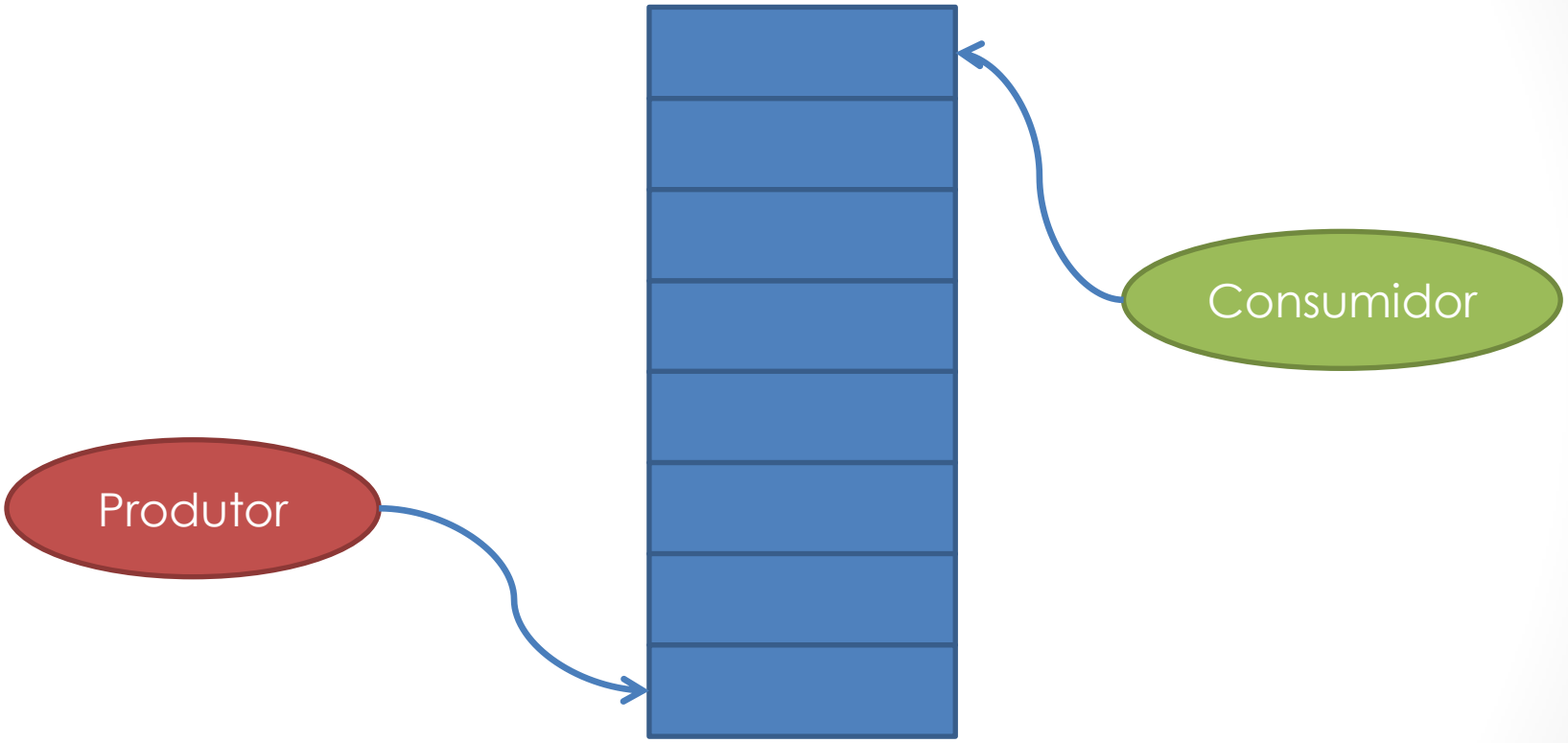
- Dependendo de como for implementado o escalonamento das threads (e do processo), a sequencia de execução das threads pode mudar
- Isso pode afetar o resultado final da execução!

- **int** sched_yield();
  - Ao ser executada, a thread voluntariamente abre mão da execução e sai do processador.
    - Isso faz com que ela volte para a fila de threads prontas
  - Ela pode voltar a ser executar normalmente se for escolhida pelo escalonador
    - Volta a executar a partir da instrução seguinte

Exercicio

- Problema do produtor/consumidor
  - Faça um programa em C que cria duas threads (além da thread principal) que simulam o comportamento de um produtor/consumidor.
    - Estas threads manipularão um buffer (vetor) global
    - Existirá uma variável inteira que aponta:
      - onde o produtor deve inserir no buffer;
      - de o consumidor onde lê desse buffer
  - Produtor
    - Escreve no buffer e atualiza o apontador
    - Deve parar de escrever se o buffer estiver cheio
  - Consumidor
    - Lê do buffer e atualiza o apontador
    - Deve parar de ler se o buffer estiver vazio
  - O produtor e o consumidor interagirão por 3 "Safras"

Exercício



Referências

- OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas operacionais**. 4. ed. Porto Alegre: Bookman, 2010. ISBN: 9788577805211.
  - **Capítulo 4**

- TANENBAUM, Andrew S.. **Sistemas operacionais modernos**. 3. ed. São Paulo: Prentice Hall, 2009. 653 p. ISBN: 9788576052371.
  - **Capítulo 2**

- SILBERCHATZ, A.; Galvin, P.; Gagne, G.; **Fundamentos de Sistemas Operacionais**, LTC, 2015. ISBN: 9788521629399
  - **Capítulo 4**

Próxima aula

- Escalonamento de processos/threads