# Linguagem de Programação II

Carlos Eduardo Batista

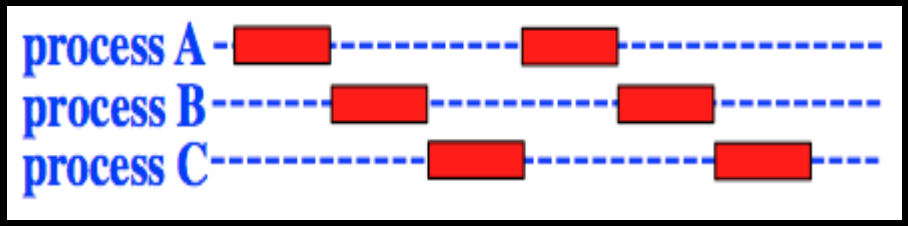
Centro de Informática - UFPB
bidu@ci.ufpb.br

# Introdução

- Concorrência
- Conceitos relacionados de arquitetura de computadores e sistemas operacionais
- Conceitos básicos de programação concorrente
- Notação
- Linguagens com suporte a programação concorrente

# Introdução

- Concorrência = Entrelaçamento ou Paralelismo
  - Entrelaçamento – processos progridem juntos, mas não usam sempre o processador




  - Paralelismo – processos progridem e executam juntos

# Introdução

- Tudo começou no projeto dos sistemas operacionais modernos
- Nos antigos SO, que só executavam um programa por vez, instruções de entrada e saída (E/S) deixavam o processador ocioso
- Multiprogramação surgiu na década de 60

# Introdução

- SO passaram a oferecer funcionalidades e bibliotecas para programação concorrente
- Suporte a concorrência em linguagens como PL/I F e ALGOL 68, depois ADA, e na década de 90 programação concorrente estaria em voga novamente em linguagens como C/C++, Java etc.

# Introdução

- Dividir um sistema em múltiplos processos ou threads não é tão fácil assim, requer **sincronismo** entre os múltiplos entes envolvidos
- Paralelizar um algoritmo não necessariamente melhora sua eficiência
- Comportamento de um programa concorrente é **dinâmico**
  - Um erro pode passar despercebido até que uma situação específica ocorra
  - *Debugging* é complexo!

# Introdução

- Experimentando concorrência
- Executando programas em *background* em sistemas Unix

```
> programa &
```

- Quando seu programa executa ele se torna um **processo**
  - Definiremos melhor mais adiante...
- Executar um programa em *background* significa que o mesmo está desassociado da janela ou terminal utilizado para executá-lo

# Introdução

- Pode-se usar & para execução de múltiplos programas em *background*
- programa1 & programa2 & programa3
  - programa3 estará em *foreground*
  - Executam em concorrência
- Teste: `ls -la & find /`
- Visualizar processos de um usuário em sistemas Unix: comando

# Introdução

- Comando **top**

```
top - 02:15:59 up 107 days, 10:11,  4 users,  load average: 0.00, 0.00, 0.00
Tasks: 158 total,   1 running, 157 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  0.2%sy,  0.0%ni, 99.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8128276k total,  2987432k used,  5140844k free,   402472k buffers
Swap: 10235896k total,     2492k used, 10233404k free,  1444264k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2480 shene     20   0 1660m 376m  40m S  1.0  4.7   2665:39 firefox
   25 root      20   0     0    0    0 S  0.3  0.0  21:59.28 ata/0
 1617 root      20   0  170m  68m  12m S  0.3  0.9 357:13.20 X
26744 shene     20   0 19204 1364 1020 R  0.3  0.0   0:00.06 top
    1 root      20   0 23464 1392 1184 S  0.0  0.0   0:02.81 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.20 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.21 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:45.56 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.21 migration/1
    7 root      20   0     0    0    0 S  0.0  0.0   0:36.84 ksoftirqd/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/1
    9 root      20   0     0    0    0 S  0.0  0.0   3:48.42 events/0
   10 root      20   0     0    0    0 S  0.0  0.0   0:26.04 events/1
   11 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cpuset
   12 root      20   0     0    0    0 S  0.0  0.0   0:00.53 khelper
   13 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
```

# Introdução

- Processos exibidos são **independentes**
- Se processos se comunicam entre si para completar uma tarefa, eles são ditos **processos cooperantes,** que requerem um planejamento de **sincronismo**
- Exemplo de cooperação entre processos, usando *pipes* Unix: **|**

```
> ls –la | more
> ps aux | grep username
```

- Redirecionar output

```
ls –la > result_ls.txt
```

# Introdução

- Outras funcionalidades de sistemas Unix relacionadas:
- Ctrl+Z: Suspende processo em *foreground* e retorna ao terminal
- bg: comando que executa processo enviado para *background* mais recente

  `> ./programa` (executa programa)

  `Ctrl + Z` (suspende programa)

  `bg` (resume programa em background)
- `fg`: traz para *foreground* processo recentemente enviado para *background*
- `kill`: finaliza processo a partir do seu PID (*process id*)

  `> kill -9 12123` (finaliza processo com PID 12123)

# Introdução à Programação Concorrrente

- Definição de concorrência
  - “Correr junto”
  - Disputa por recursos
  - Cooperação
- Programação concorrente
  - Utilização de dois ou mais processos que cooperam para solucionar um determinado problema

# Introdução à Programação Concorrrente

- Programação concorrente
  - A cooperação dos processos se dá por meio da comunicação entre eles.
    - Compartilhamento de memória
    - Troca de mensagens
  - Necessidade de sincronização
    - Exclusão mútua
    - Sincronização por condição

# Introdução à Programação Concorrrente

- Sincronização por exclusão mútua
  - Visa garantir que os trechos de código em cada thread que acessam objetos compartilhados não sejam executados ao mesmo tempo.
- Sincronização por condição
  - Visa garantir que uma thread seja retardada enquanto uma determinada condição lógica da aplicação não for satisfeita.

# Introdução à Programação Concorrrente

- Com o advento das Redes de Computadores, surgem os sistemas distribuídos: programas que executam suas partes em computadores fisicamente separados que se comunicam via troca de mensagens.

# Motivação

- Adaptar a estrutura lógica de um problema (Ex.: Servidores Web).
- Lidar com dispositivos independentes (Ex.: S.O.).
- Aumentar o desempenho das aplicações (Ex.: aplicações de Big Data, SETI@Home).
- Arquiteturas de computadores paralelos
  - Avanços mais recentes das Arquitetura de Computadores têm implicado muito mais no incremento de paralelismo (multiprocessadores ou multicores) do que no aumento velocidade (*clock*).

# Introdução

- Exercício: executando programas concorrentes
- Programa 1: lê um valor inteiro da linha de comando e exibe esta quantidade de linhas (strings separadas por \n)
- Programa 2: lê um valor inteiro e exibe estas linhas concatenando uma mensagem
- Executar: Programa_1 | Programa_2
  - (output exemplo)
  - Prog2: Lendo 3 Linhas de Prog1
  - Prog2: Linha 1 de Prog1
  - Prog2: Linha 2 de Prog1
  - Prog2: Linha 3 de Prog1

# Referências


- Notas de Aula do Prof. Bruno Pessoa
- Andrews, G. *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison-Wesley, 2000.
- Rosseto, S. , “Cap. I: Introdução e histórico da programação concorrente”. Disponível em: <http://www.dcc.ufrj.br/~silvana/compconc/cap-1.pdf>.