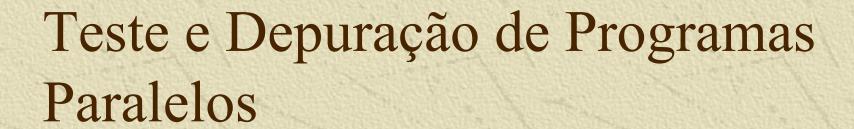
Teste e Depuração de Programas Concorrentes

Fábio Lima Santos Fernando Vacari Juliano Rosa



Teste e Depuração de Programas Concorrentes

- * Introdução
- ** Teste de Programas Paralelos
- * Depuração de Programas Paralelos
- ** Problemas com Teste e Depuração
- * Erros mais comuns e prevenção
- * PVM e MPI
- * Ferramentas Experimentais
- * Conclusão
- * Bibliografia



"First, the bad news. Adding printf() calls to your code is still a state-of-the-art methodology."

Extraído do site http://www.netlib.org/pvm3/book



- * Depuração de programas sequenciais: arte difícil.
- * Depuração de programas paralelos: obrigação dolorosa.
- ** Programas paralelos apresentam os mesmos problemas de programas seqüenciais e vários outros.



- *Grande quantidade de pesquisa, mas poucos resultados.
- ** Duas frentes de pesquisa e desenvolvimento:
 - Desenvolvimento de ferramentas para depuração e teste.
 - Desenvolvimento de um modelo ou arquitetura que evite que o programador cometa erros.



* Situação atual:

- Ferramentas muito específicas e dependentes da plataforma.
- •Diversos problemas ainda sem solução.
- Maioria em fase experimental.



- * "Todo programa não trivial contém erro."
- ** Erros: quaisquer ocorrências ou eventos que venham a fazer com que o software não apresente o comportamento ou resultado desejado.
- ** Teste: método para se descobrir se existem erros em um programa.
- ** Testes bem sucedidos implicam garantia de qualidade de software.



- ** Programas sequenciais:
 - Resultados determinísticos.
 - Vários métodos de teste consolidados.
 - Grande quantidade de pesquisa na área.
 - •Já existem bastantes estudos teóricos baseados em modelos matemáticos e estatísticos.
 - Percurso muito grande a ser percorrido.



** Programas Paralelos:

- Tem esbarrado em problemas pouco explorados.
- Abordagens sequenciais são, em geral, ineficazes.
- Resultados não-determinísticos.



* Erros:

- Erros comuns em programas seqüenciais: loops infinitos, manipulação de memória, etc...
- Novos erros:
 - Race Condition.
 - · Deadlocks.
 - · Livelocks.



** Race Condition:

- •O programa é executado, porém o resultado depende da ordem de execução dos processos.
- •Em geral, ocorrem por causa de má proteção de memória compartilhada.



* Deadlock:

- Processos ficam parados aguardando algum evento ou condição.
- Geralmente, por falha na utilização de mecanismos de sincronização.



* Livelock:

- Os processos entram em um ciclo onde não conseguem realizar nenhuma tarefa completamente e nem sair deste ciclo.
- Falha no processo de sincronização.
- Muito difícil de detectar.



- **Programas sequenciais: dada uma entrada, uma única saída é obtida (estando correta ou não).
- ** Programas paralelos: dada uma entrada, pode-se obter saídas distintas (geralmente em programas que contém erros), dependendo do estado do sistema.

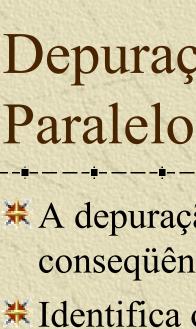


** Problemas:

- descobrir quais os estados do sistema para os quais o programa não funciona corretamente.
- Efeito de Intrusão: a inserção de diretivas de depuração no código concorrente pode modificar o estado do sistema, escondendo possíveis erros.

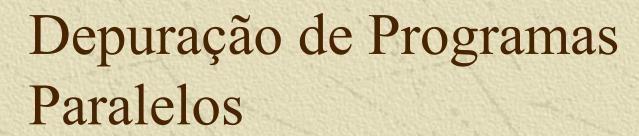


- **Basicamente, testar um sistema consiste em se desenvolver um caso de teste, ou seja, uma entrada e uma saída esperada do sistema.
- *Em programas paralelos, o mesmo teste deve ser executado para cada possível estado do sistema.
- ** Como determinar todos os possíveis estados do sistema?
- ** Como reproduzir cada estado do sistema?

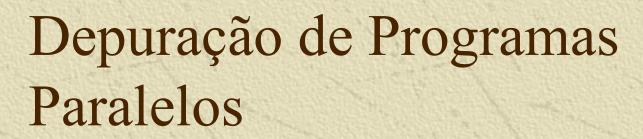


Depuração de Programas Paralelos

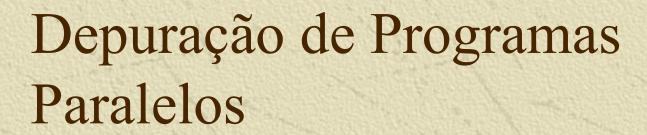
- *A depuração de programas paralelos é uma consequência de testes bem sucedidos.
- * Identifica onde estão os erros encontrados na fase de teste.
- *Não existem métodos prontos para se descobrir onde e porque um código não está funcionando.
- * Existem técnicas e ferramentas que auxiliam o programador nesta tarefa.



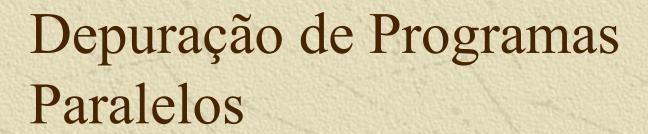
- * Depuração Tradicional:
 - Um depurador para cada processo.
 - Cada processo é analisado separadamente e concorrentemente.
 - Depuração de Saída: exibe passo a passo o valor das variáveis e estado do sistema.
 - Tracing: o valor das variáveis é visualizado em certos pontos da execução.



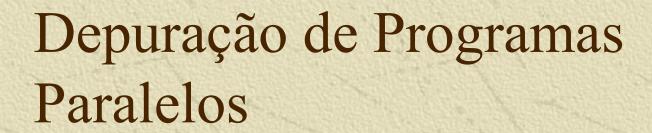
- * Depuração Tradicional:
 - •Breakpoints: permite que em determinado ponto da execução de um processo, o valor das variáveis seja visualizado e alterado.
 - Controle de Execução: permite determinar a ordem de execução dos eventos, alterando o estado do sistema.



- * Depuração Baseada em Eventos:
 - •Os eventos são as interações entre os processos ou dentro dos processos.
 - Pode armazenar o histórico dos eventos, podendo posteriormente reproduzir o estado do sistema, caso algum erro ocorra.
 - Browsing
 - Replay
 - Simulação

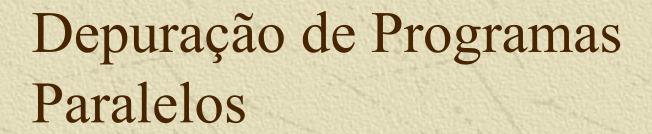


- # Fluxo de Controle:
 - Representação de toda a comunicação e compartilhamento entre processos de um sistema.
 - Textos, diagramas, animações, etc...
 - Visa facilitar a visualização do que está ocorrendo ou já ocorreu.



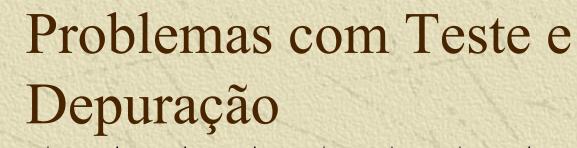
* Análise Estática:

- Descobrir falhas estruturais ao invés de identificar falhas funcionais.
- Eliminação de erros antes da execução de testes.
- Muito utilizada para detecção de erros comuns: erros de sincronização e nas operações com dados compartilhados.
- Não produz o Efeito de Intrusão.
- Altamente complexo.
- Alto custo computacional.



* Monitoração:

- Análise do estado e funcionamento do programa em sua iteração com situações reais.
- *Geração de logs ou relatórios.
- •São monitoradas informações como o status dos processos e nós, tráfego de informações e as transações do sistema.
- •Em geral, existem ferramentas para análise destes relatórios.



- *As primeiras ferramentas não obtiveram sucesso pois detectavam muitas falhas e erros que não existiam na realidade.
- ** Fazia com que os programadores não as utilizassem.
- ** Utilizava Análise Estática indiscriminadamente.
- *As ferramentas mais atuais aplicam análises para verificar se as possíveis falhas podem causar erros no resultado final.



- * Existem erros praticados pelos programadores que são comuns.
- *O conhecimento destes erros é uma grande ajuda no desenvolvimento de técnicas e ferramentas para teste e depuração.
- *A seguir, tem-se alguns exemplos de erros comuns encontrados em programação concorrente.



** Bibliotecas:

- •O uso de bibliotecas é um ponto de introdução de erros por programadores inexperientes.
- Algumas bibliotecas, as chamadas "threadsafe", possuem proteção para memória, outras não.
- •Programadores experientes fazem testes exaustivos mesmo usando bibliotecas "threadsafe".



- * Assumindo ordem de execução:
 - Programadores inexperientes esperam que os comandos sejam executados na ordem em que estão escritos (resquício da programação estruturada).
 - A seguir veremos um exemplo que possui um resultado não-determinístico.

•Exemplo:

```
#include <stdio.h>
#include <pthread.h>
#define N 4
void *hello (void*);
int main() {
   int j;
   pthread t tid[N];
   for (j = 0; j < N; j++)
        pthread create (&tid[j], NULL, hello, (void *)&j);
   for (j = 0; j < N; j++)
        pthread join (tid[j], NULL);
void *hello (void *my id) {
printf ("Hello World from thread %d\n", *(int *)my id);
   return NULL;
```



- * Variáveis Mutex em escopo incorreto:
 - Variáveis mutex declaradas em escopos incorretos fazem com que cada thread tenha um mutex diferente.
 - Dentro destas condições nunca se terá a exclusão mútua desejada.
 - A seguir tem-se um exemplo onde o mutex atua apenas dentro da função exibida, que é executada independentemente em cada thread.

•Exemplo:

```
int counter = 0; /* contador compartilhado */
void increment counter (void) {
   pthread mutex t *lock;
   lock = (pthread mutex t *)
             malloc( sizeof(pthread mutex t) );
   pthread mutex init (lock, NULL);
   pthread mutex lock (lock);
      counter++;
   pthread mutex unlock (lock);
```



- ** Perda de sinais:
 - •É de total responsabilidade do programador sincronizar o envio do sinal de condição e o recebimento do mesmo.
 - •Se um thread envia a mensagem a outro em um momento em que este não a está esperando, o sinal é perdido.
 - •Em algumas arquiteturas este sinal não é perdido.



- * Hierarquia de Locks:
 - Geralmente, programas possuem mais de um mutex.
 - •É desaconselhável executar "lock" em um mutex sem executar "unlock" em outro mutex usado anteriormente.
 - •O exemplo a seguir ilustra este problema.

•Exemplo:

```
#include <stdio.h>
#include <pthread.h>
void *thread1 (void*);
void *thread2 (void*);
pthread mutex t lock1 = PTHREAD MUTEX INITIALIZER;
pthread mutex t lock2 = PTHREAD MUTEX INITIALIZER;
int main() {
   pthread t tid1, tid2;
   pthread create (&tid1, NULL, thread1, NULL);
   pthread create (&tid2, NULL, thread2, NULL);
   pthread join (tid1, NULL);
   pthread join (tid2, NULL);
```

Exemplo (continuação): void *thread1 (void *arg) { pthread mutex lock (&lock1); printf ("Thread 1 holding first lock\n"); pthread mutex lock (&lock2); printf ("Thread 1 holding second lock\n"); pthread mutex unlock (&lock2); pthread mutex unlock (&lock1); return NULL; void *thread2 (void *arg) { pthread mutex lock (&lock2); printf ("Thread 2 holding second lock\n"); pthread mutex lock (&lock1); printf ("Thread 2 holding first lock\n"); pthread mutex unlock (&lock1); pthread mutex unlock (&lock2); return NULL;



- ** Threads interrompidos:
 - Causas:
 - Um thread é terminado inesperadamente;
 - O sistema operacional não consegue alocar o número de threads exigido.
 - •Resultado:
 - · Deadlock.
 - Resultado incorreto.



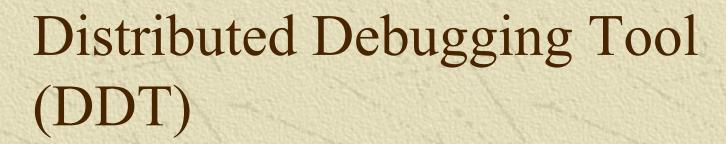
- *Em PVM existem alguns elementos que podem auxiliar no processo de depuração:
 - PvmDebugMask
 - *PvmAutoErr
- * Existem, tanto para PVM quanto para MPI, ferramentas que auxiliam nas tarefas de teste e depuração.



- # Englobam várias operações:
 - Análise estática
 - **Teste**
 - Depuração
- *A grande maioria está em fase experimental e não apresentam a eficiência e o desempenho desejados.
- * Existem também as ferramentas de monitoração de programas tanto paralelos quanto seqüenciais.



- * Ferramenta produzida pela Etnus.
- ** Análise de código e depuração (não permite realização de testes automáticos).
- ** Suporte para teste e depuração de programas:
 - **PVM**
 - **+**MPI
 - *OpenMP
 - ◆Threads em Fortran, C e C++
- ** Considerada a melhor ferramenta para teste e depuração existente para Linux e Unix.



- ** Produzida pela Streamline Computing.
- * Suporte para:
 - **•**MPI
 - •Fortran 77 e 90
 - **♦**C e C++
- * Fácil visualização do status de cada processo.
- * Depuração linha a linha.
- * Existem versões para Linux, Solaris e Alpha.



- * Existem ferramentas experimentais desenvolvidas em projetos de pesquisa.
- * Estes projetos estudam novos métodos principalmente na detecção de erros reais.
- ** Poucos tratam de MPI e PVM, a maioria está ligada à implementação de threads.



- ** Teste e depuração é uma etapa essencial para se garantir um software de qualidade.
- * Ferramentas ineficazes e pouco numerosas.
- É muito importante que haja novas pesquisas na área e, que as já em andamento continuem para que se obtenha bons métodos de detecção, localização e prevenção de erros.



- NETO, J. C. da C. Teste Estrutural Baseado em Fluxo de Dados de Programas Concorrentes, ICMC USP.
- SANTANA, Regina Helena Carlucci; SANT'ANA, Tomás Dias; ASTRAL Ambiente de Simulação e Teste de pRogramas parALlos, ICMC USP e UNIFENAS.
- Etnus em http://www.etnus.com visitado em junho de 2003
- IGLINSKI, Paul Using a Template-Based Parallel Programming Environment to Eliminate Errors, Department of Computing Science, University of Alberta, Canadá.
- PVM em http://www.csm.ornl.gov/pvm/pvm_home.html visitado em junho de 2003
- MPI em http://www-unix.mcs.anl.gov/mpi/ visitado em junho de 2003
- GABB, Henry Common Concurrent Programming Errors, Linux Magazine, março de 2002 em http://www.linux-mag.com/2002-03/concurrent_01.html visitado em junho de 2003