Programação Concorrente

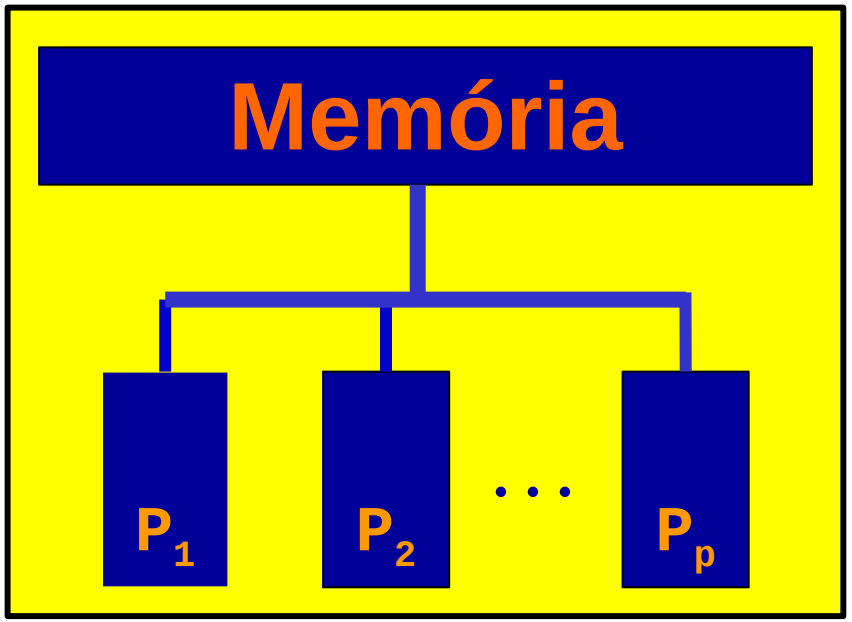
POSIX Threads (Pthreads)

André Luis Martinotto

UCS UNIVERSIDADE DE CAXIAS DO SUL

Threads

- Dois ou mais processadores compartilham uma memória principal comum (Multiprocessadores)
- Qualquer processo ou processador pode ler ou escrever qualquer palavra na memória compartilhada



Threads

- A forma mais comum de exploração de paralelismo em máquinas com esse tipo de memória é o uso de threads.
- Uma thread é um fluxo de execução de instruções;
- Um processo tradicional tem um único fluxo de execução.
- Sistemas Operacionais oferecem recursos para que um processo contenha múltiplos fluxos de execução

Threads

- No UNIX, uma thread:
  - Existe no interior de um processo e se utiliza dos recursos alocados para ele
  - Compartilha os recursos do processo ao qual pertence com outras threads do mesmo
  - Morre quando o processo a que pertence morre

Threads

- Cada processo possui atributos que são compartilhados por suas threads.
  - PID, PPID, GID e UID
  - Variáveis de ambiente
  - Diretório corrente
  - Código fonte
  - Alguns registradores especiais
  - Área de dados (Globais + Heap)
  - Descritores de arquivos

Threads

- Cada thread é composta por:
  - Um identificador
  - Um contador de programa
  - Um conjunto de registradores
  - Uma pilha

Threads

- Todas as threads de um mesmo processo executam no mesmo espaço de endereçamento
- Num processo multi-threaded, existe mais de um ponto de execução simultaneamente
  - Multithreading é uma ferramenta poderosa para tratar problemas naturalmente concorrentes
    - Programador não necessita tratar múltiplas atividades em um único fluxo de execução;

Threads X Processos

- A criação e a sincronização das threads são mais rápidas
- A comunicação entre threads é mais eficiente, por causa do espaço de endereçamento compartilhado
- Melhor eficiência em arquiteturas SMP

Bibliotecas de Threads

- Existem três modelos básicos utilizados na implementação de bibliotecas de threads:
  - 1:1 (one-to-one)
  - N:1 (many-to-one)
  - M:N (many-to-many)
- Esses diferenciam-se pelo mecanismo de escalonamento utilizado para a alocação de threads ao processador.

Modelo 1:1 (one-to-one)

- Suportadas diretamente pelo sistema operacional;
  - Recebem o nome de threads de sistema;
- São escalonadas da mesma forma que processos;
- Atraentes para ambientes multiprocessados
  - Diferentes threads podem rodar em processadores diferentes;
- Exemplo: POSIX threads em Linux

Modelo N:1 (many-to-one)

- Suporte a threads não é oferecido pelo sistema operacional;
- Utilização de bibliotecas de threads usuário;
- Executam ao mesmo nível da aplicação
  - O escalonamento da thread é realizado quanto o processo for escalonado pelo sistema operacional;
- Não exploram por completo o paralelismo em arquiteturas multiprocessadas;
- A manipulação de threads usuário é menos onerosa que a manipulação de threads de sistema
- Exemplo: existem implementações para Windows e Minix

Modelo MxN (many-to-many)

- Mescla as características de ambos os modelos (1:1 e N:1);
- Cada processo pode comportar N threads de sistema com M threads usuário
- Exemplo: Solaris

POSIX Threads

- É uma interface de manipulação de threads padronizada em 1995 pelo IEEE (IEEE POSIX 1003.1c)
- POSIX threads -> Pthreads
- Pthreads foi definido como um conjunto de tipos e procedimentos em C
  - Definidos em pthreads.h

POSIX Threads

- Do ponto de vista do programador:
  - Uma thread é uma função que é executada de forma independente do seu programa principal
    .
- Cuidados:
  - Nem todas as funções das bibliotecas foram projetadas para trabalhar com threads (thread-safe)
  - Assuma que a função NÃO é thread-safe

POSIX Threads

- Contem mais de 60 funções
- Incluir sempre pthreads.h
- O padrão é definido apenas para a linguagem C
-

POSIX Threads

- É dividida em 3 grandes categorias:
  - Gerenciamento de threads:
    - Criação, configuração, escalonamento
  - Mutexes:
    - Exclusão mútua
  - Variáveis condicionais
    - Comunicação entre threads que compartilham mutexes

Usando Pthreads

- Para compilar:
  - gcc nome.c -o nome -lpthread
-
- Para executar:
  - ./nome

Criação de Threads

- Para criar threads:
-
  - int pthread_create (pthread_t * thread, pthread_attr_t *attr, void * (*start_routine)(void *), void * arg);
  .
- Valor de retorno: 0 se funcionar, ou um valor indicando erro, caso contrário

Criação de Threads

- **pthread_create:**
  - Para criar threads;
-
- **Sintaxe:**
  - int pthread_create (pthread_t * thread, pthread_attr_t *attr, void * (*rotina)(void *), void * arg);
  .
- Valor de retorno: 0 se funcionar, ou um valor indicando erro, caso contrário

Criação de Threads

- Rotina é a função em C onde será iniciada a nova thread.
-
- Apenas o parâmetro arg é passado para a nova thread.
  - Caso seja necessário passar mais de um parâmetro é necessário criar uma struct;

Criação de Threads – Exemplo 1

```
#include <pthread.h>

void *funcao (void *args){
        printf("Hello World\n");
}

int main() {
        pthread_t tid;
        pthread_create( &tid, NULL, funcao , NULL );
}
```

Criação de Threads – Exemplo 2

```
#include <pthread.h>

#define NTHREADS 5

void *funcao (void *args){
        printf("Hello World Thread %i\n", args);
}

int main() {
        pthread_t tid[NTHREADS];
        int i;
        for ( i=0 ; i< NTHREADS; i++ )
                pthread_create( &tid[i], NULL, funcao, (void *)i );
}
```

Joining Threads

- **pthread_join**
  - "Joining" é uma maneira de se obter sincronização entre threads.
  - Para que uma thread fique bloqueada até que uma outra termine:
-
- **Sintaxe:**
- int pthread_join (pthread_t thread, void **status);
  - thread: indica a thread a ser aguardada
  - status: retorno da função executada pela thread

Joining Threads – Exemplo

```
#include <pthread.h>

#define NTHREADS 5

int fatorial(int n){
        if (n<2)
                return(1);
        else return( n * fatorial(n-1));
}

void *funcao (void *args){
        int fat = fatorial((int)args);
        return (void *)fat;
}
```

Joining Threads – Exemplo

```
int main() {
    int n,p, fatp, fatn, fatnp;
    pthread_t thidN, thidP;
    printf("Digite o valor de n,p: ");
    scanf("%i %i",&n,&p);
    printf("%i %i\n",n,p);

    pthread_create(&thidN,NULL,funcao,(void *)n);
    pthread_create(&thidP,NULL,funcao,(void *)p);

    fatnp = fatorial(n-p);

    pthread_join(thidN,(void *)&fatn);
    pthread_join(thidP,(void *)&fatp);

    printf("Binomio: %f\n", (float)fatn/(fatp*fatnp));
}
```

Identificador da Thread

- **pthread_self**:
  - Retorna o identificador da thread corrente;
  - .
- **Sintaxe:**
    - pthread_t pthread_self (void)

Identificador da Thread

```
#include <pthread.h>
#define NTHREADS 5

void *funcao (void *args){
        printf("Hello World %i\n", pthread_self());
}

int main() {
        pthread_t tid[NTHREADS];
        int i;
        for ( i=0 ; i< NTHREADS; i++ )
                pthread_create( &tid[i], NULL, funcao, (void *)i );

        for ( i=0 ; i< NTHREADS; i++ )
                pthread_join( tid[i],NULL);
}
```

Término de uma Thread

- Uma thread termina quando:
  - A thread retorna da função que a originou
  - A thread chama pthread_exit
  - A thread é cancelada por outra thread através da função pthread_cancel
  - O processo inteiro termina

Término de uma Thread

- **pthread_exit**
  - Termina a execução da thread corrente.
  - .
- **Sintaxe:**
    - int pthread_exit(void *status)
  - .
  - status: retorno da função executada pela thread

Término de uma Thread – Exemplo 1

```
#include <pthread.h>
#define NTHREADS 5

void *funcao (void *args){
        if ( (int)args & 1 )
                pthread_exit(NULL);
        printf("Hello World %i\n", args);
}

int main() {
        pthread_t tid[NTHREADS];
        int i;
        for ( i=0 ; i< NTHREADS; i++ )
                pthread_create( &tid[i], NULL, funcao, (void *)i );

        for ( i=0 ; i< NTHREADS; i++ )
                pthread_join( tid[i],NULL);
}
```

Término de uma Thread

- **pthread_cancel**
  - Cancela uma determinada thread
  - .
- **Sintaxe:**
    - int pthread_cancel (pthread_t thid)
  - thid: identificador da thread a ser cancelada;

Término de uma Thread – Exemplo 2

```
#include <pthread.h>

void *funcao (void *args){
        while(1)
                printf("Hello World\n");
}

int main() {
        pthread_t tid;
        pthread_create( &tid, NULL, funcao, NULL );
        sleep(1);
        pthread_cancel(tid);
        pthread_join(tid,NULL);
}
```

Exclusão Mútua

- Mutex é uma abreviação de "mutual exclusion" (exclusão mútua)
- Variáveis do tipo Mutex são a principal forma para a proteção de regiões críticas
- Uma variável do tipo mutex é um lock que protege dados compartilhados pelas threads.
- O princípio básico é que apenas uma thread pode ter efetuado um lock em uma variável do tipo mutex em um dado instante.

Exclusão Mútua

- Mesmo que diversas threads tentem efetuar o lock, apenas uma delas será bem sucedida.
- Nenhuma outra thread poderá efetuar o lock antes que a primeira thread o libere.
- Aquelas que não conseguiram efetuar o lock ficam bloqueadas na chamada da função de lock;
-
- Variáveis do tipo mutex são do tipo **pthread_mutex_t**;
-

Exclusão Mútua

- **pthread_mutex_init**
  - Inicializa um mutex
  -
- **Sintaxe:**
  - int pthread_mutex_init (pthread_mutex_t *mutex, pthread_mutexattr_t *attr)
  - mutex: é o mutex
  - attr: define se o valor é lock ou unlock para o mutex
    - Default é unlock (selecionada attr NULL)

Exclusão Mútua

- **pthread_mutex_lock**
  - Operação de lock no mutex
  -
- **Sintaxe:**
  - int pthread_mutex_lock (pthread_mutex_t *mutex)
  -
  -
- **pthread_mutex_unlock**
  - Operação de lock no mutex
  -
- **Sintaxe:**
  - int pthread_mutex_unlock (pthread_mutex_t *mutex)

Exclusão Mútua – Exemplo 1

```
#include <pthread.h>

#define NTHREADS 5

pthread_mutex_t mutex;

void *funcao (void *args){
        pthread_mutex_lock(&mutex);
        printf("Hello World %i\n",args);
        sleep(5);
        pthread_mutex_unlock(&mutex);
}
```

Exclusão Mútua – Exemplo 1

```
int main() {
    pthread_t tid[NTHREADS];
    int i;

    pthread_mutex_init(&mutex,NULL);

    for ( i=0 ; i< NTHREADS; i++ )
        pthread_create( &tid[i], NULL, funcao, (void *)i);

    for ( i=0 ; i< NTHREADS; i++ )
        pthread_join( tid[i], NULL);

}
```

Exclusão Mútua – Exemplo 2

```
#include <pthread.h>
#define NTHREADS 5
#define TAM 100

pthread_mutex_t mutex;
int vetor[TAM], prodtotal=0;

void *funcao (void *args){
    int ini, fim, parte, i, prodparc=0;
    ini = (int)args * (TAM/NTHREADS);
    fim = ini + (TAM/NTHREADS);
    for ( i=ini; i<fim ; i++ )
        prodparc += vetor[i] * vetor[i];

    pthread_mutex_lock(&mutex);
    prodtotal += prodparc;
    pthread_mutex_unlock(&mutex);
}
```

Exclusão Mútua – Exemplo 2

```
int main() {
    pthread_t tid[NTHREADS];
    int i;
    pthread_mutex_init(&mutex,NULL);

    srand(time(NULL));
    for(i=0;i<TAM;i++)
        vetor[i] = rand()%10;

    for ( i=0 ; i< NTHREADS; i++ )
        pthread_create( &tid[i], NULL, funcao, (void *)i);

    for ( i=0 ; i< NTHREADS; i++ )
        pthread_join( tid[i], NULL);

    printf("Total: %i\n",prodtotal);
}
```

Variáveis de Condição

- Em determinados casos um thread só deve entrar em uma sessão crítica se:
  - Obter o direito de acesso exclusivo com o lock
  - Se uma determinada condição for atendida;
  -
- Isso é realizado através de um mecanismo chamado de variáveis de condição;
  - Deve ser obrigatoriamente utilizado com mutex (a variável de condição é compartilhada pelas threads);

Variáveis de Condição

- Uma variável de condição pode ser construída a partir do tipo: **pthread_cond_t**
- **pthread_cond_init**
  - Inicialização de uma variável de condição
- **Sintaxe:**
  - int pthread_cond_init (pthread_cond_t *cond, pthread_condattr_t *cond_attr);
- cond: variável de condição
- cond_attr: indica se a condição inicial encontra-se satisfeita (default -NULL) ou não

Variáveis de Condição

- A manipulação das variáveis de condição se dá através das seguintes primitivas:
  - pthread_cond_wait
  - pthread_cond_signal
  - pthread_cond_broadcast

Variáveis de Condição

- **pthread_cond_wait**
  - Faz com que a thread seja bloqueada na espera de uma sinalização;
  - **Importante:** para evitar deadlock no momento que a thread é bloqueada no wait o mutex associando é liberado
  .
- **Sintaxe:**
  - pthread_cond_wait(pthread_cond_t *cond, pthread_mutex *mutex);
  .

Variáveis de Condição

- **pthread_cond_signal**
  - Envia um sinal para apenas uma das threads bloqueadas pela variável de condição
  .
- **Sintaxe:**
  - pthread_cond_signal(pthread_cond_t *cond);
  .

Variáveis de Condição

- **pthread_cond_broadcast**
  - Envia um sinal para todas as threads bloqueadas pela variável de condição
  .
- **Sintaxe:**
  - pthread_cond_broadcast(pthread_cond_t *cond);
  .

Variáveis de Condição

```
#include <pthread.h>
#define NTHREADS 5

pthread_mutex_t mutex;
pthread_cond_t cond;
int barreira=0;

void *funcao (void *args){
        sleep((int)args*5);
        pthread_mutex_lock(&mutex);
        barreira++;
        if ( barreira < NTHREADS ){
                printf("Thread %i foi bloqueada\n",args);
                pthread_cond_wait(&cond,&mutex);
        }
        printf("Thread %i foi desbloqueada\n",args);
        pthread_cond_broadcast(&cond);
        pthread_mutex_unlock(&mutex);
}
```

Variáveis de Condição

- int main() {
  - pthread_t tid[NTHREADS];
  - int i;
-
  - pthread_mutex_init(&mutex,NULL);
  - pthread_cond_init(&cond,NULL);
-
  - for (i=0 ; i< NTHREADS; i++)
    - pthread_create(&tid[i], NULL, funcao, (void *)i);
-
  - for (i=0 ; i< NTHREADS; i++)
    - pthread_join(tid[i], NULL);
- }

Problema dos Leitores e Escritores

- Este problema modela o acesso a uma grande base de dados, por exemplo, um sistema de passagens de uma companhia aérea;
- Com muitos processos competindo pelo direito de ler e escrever.
- É aceitável que haja mais de um processo lendo a base de dados ao mesmo tempo
- Mas se um processo estiver escrevendo na base de dados, nenhum outro processo, nem mesmo os leitores, poderão ter acesso a ela enquanto o escritor não terminar.

Problema dos Leitores e Escritores

```
#include <pthread.h>
#define NESCRITORES 1
#define NLEITORES 3
pthread_mutex_t mutex;
pthread_mutex_t database;
int nleitores=0;

void *esc (void *args){
        int id = (int)args;
        while(1){
                pthread_mutex_lock(&database);
                printf("Escritor %i Acessando a base\n",id);
                sleep(rand()%10);
                printf("Escritor %i Saindo da base\n",id);
                pthread_mutex_unlock(&database);
                sleep(rand()%10);
        }
}
```

Problema dos Leitores e Escritores

```
void *leit (void *args){
    int id = (int) args;
    while(1){
        pthread_mutex_lock(&mutex);
        nleitores++;
        if (nleitores==1)
            pthread_mutex_lock(&database);
        pthread_mutex_unlock(&mutex);
        printf("Leitor %i Acessando a base\n",id);
        sleep(rand()%10);
        printf("Leitor %i Saindo da base\n",id);
        pthread_mutex_lock(&mutex);
        nleitores--;
        if (nleitores==0)
            pthread_mutex_unlock(&database);
        pthread_mutex_unlock(&mutex);
        sleep(rand()%10);
    }
}
```

Problema dos Leitores e Escritores

```
int main() {
    int i;
    pthread_t leitores[NLEITORES],escritores[NLEITORES];

    srand(time(NULL));
    pthread_mutex_init(&mutex,NULL);
    pthread_mutex_init(&database,NULL);

    for ( i =0 ;i < NESCRITORES; i++)
        pthread_create( &escritores[i], NULL, esc, (void *)i);
    for ( i =0 ;i < NLEITORES; i++)
        pthread_create(&leitores[i],NULL,leit,(void *)i);

    for ( i =0 ;i < NESCRITORES; i++)
        pthread_join(escritores[i],NULL);
    for ( i =0 ;i < NLEITORES; i++)
        pthread_join(leitores[i],NULL);
}
```