Sockets - Múltiplas Conexões Usando: THREADS e SELECT

Bruno Pereira

Universidade Federal de Minas Gerais

bruno.ps@live.com

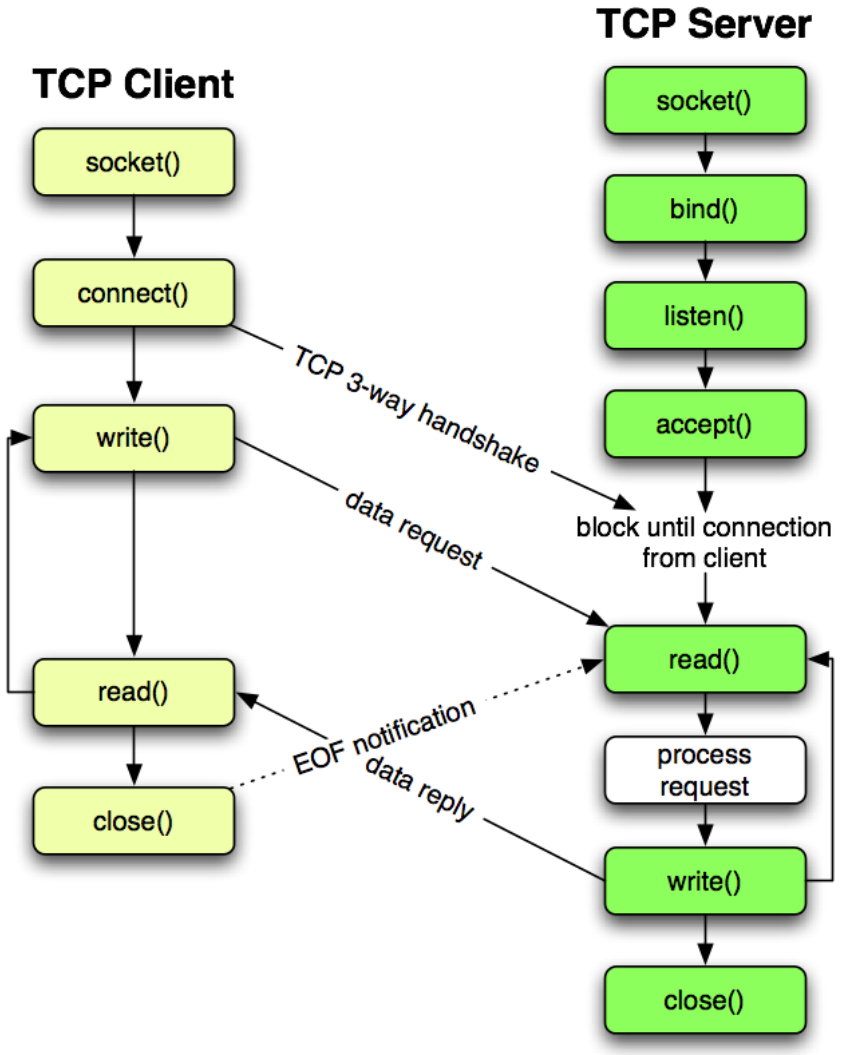
27 de setembro de 2016

Roteiro

Introdução

Relembrando

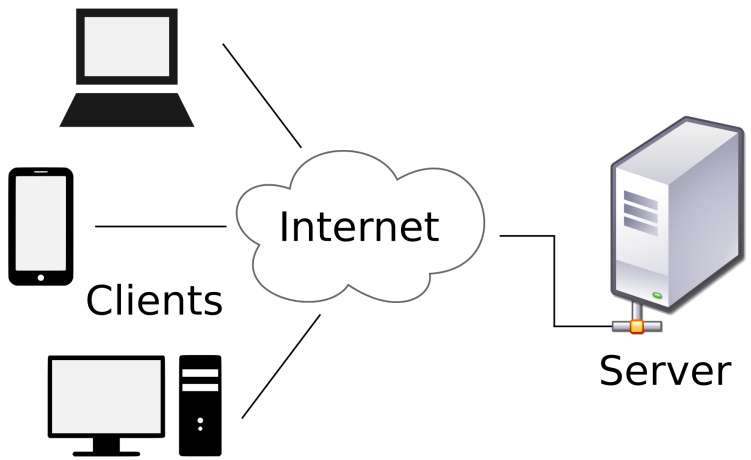
- Até o momento...
  - Sabemos trocar dados entre:
    - **UM** Cliente e **UM** Servidor
  - Tratar conexões IPv*
  - Transporte: STREAM vs DGRAM
- Agora queremos...
  - Manipular conexões simultaneamente!
- Como fazer ?
  - Threads
  - Select
  - Processos
  - libevent



Introdução

Relembrando

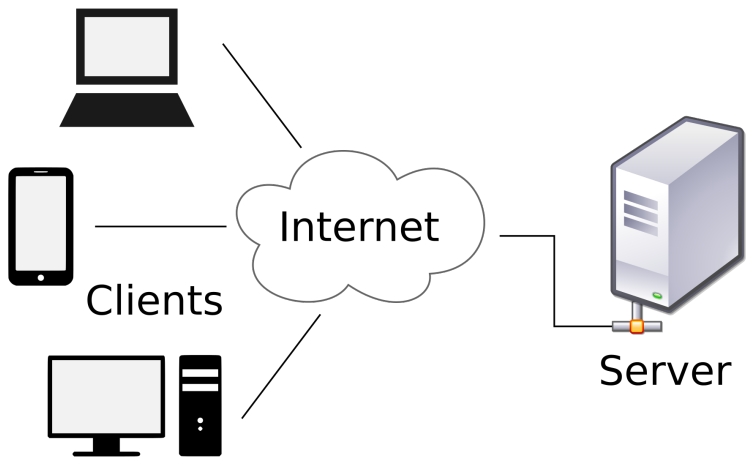
- Até o momento...
  - Sabemos trocar dados entre:
    - **UM** Cliente e **UM** Servidor
  - Tratar conexões IPv*
  - Transporte: STREAM vs DGRAM
- Agora queremos...
  - Manipular conexões simultaneamente!
- Como fazer ?
  - Threads
  - Select
  - Processos
  - libevent



Introdução

Relembrando

- Até o momento...
  - Sabemos trocar dados entre:
    - **UM** Cliente e **UM** Servidor
  - Tratar conexões IPv*
  - Transporte: STREAM vs DGRAM
- Agora queremos...
  - Manipular conexões simultaneamente!
- Como fazer ?
  - **Threads**
  - **Select**
  - Processos
  - libevent



Objetivos

- Ao final dessa aula você será capaz de:
  - Entender o uso de threads
  - Entender o uso do *Select*
- Criar um programa **SERVIDOR** que aceita conexão de vários clientes
  - Usando *Threads*
  - Usando *Select*
  - Usanto libevent
- Vamos criar um **SERVIDOR** que cria uma **THREAD** para cada cliente
  - Cada threads cuidará da lógica/execução de cada cliente individualmente

Objetivos

- Ao final dessa aula você será capaz de:
  - Entender o uso de threads
  - Entender o uso do *Select*
- Criar um programa **SERVIDOR** que aceita conexão de vários clientes
  - Usando *Threads*
  - Usando *Select*
  - Usanto libevent
- Vamos criar um **SERVIDOR** que cria uma **THREAD** para cada cliente
  - Cada threads cuidará da lógica/execução de cada cliente individualmente

O que é Thread?

- Antes de definir threads vamos entender o que é um PROCESSO.
- Em um processo existe uma thread (uma linha de execução)

O que é Thread?

- Antes de definir threads vamos entender o que é um PROCESSO.
- Em um processo existe uma thread (uma linha de execução)

PROCESSO

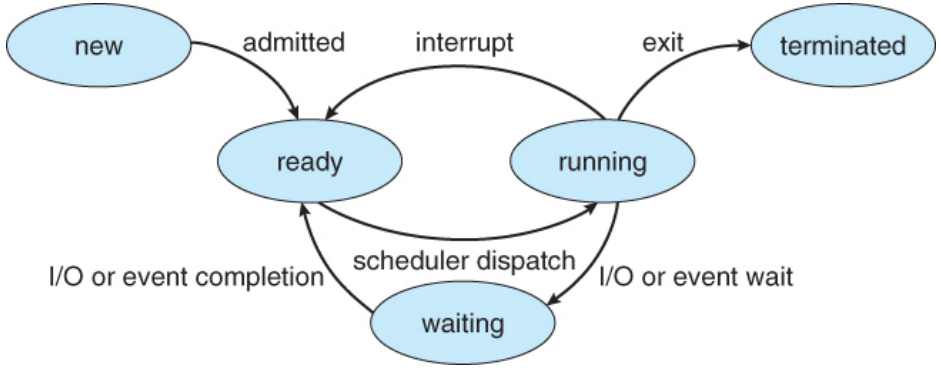
- Um processo é uma instância de um programa que está em *execução*
- Está em um dos seus 5 estados
- Possui vários atributos no seu *Process Control Block*

O que é Thread?

- Antes de definir threads vamos entender o que é um PROCESSO.
- Em um processo existe uma thread (uma linha de execução)

PROCESSO

- Um processo é uma instância de um programa que está em *execução*
- Está em um dos seus 5 estados
- Possui vários atributos no seu *Process Control Block*

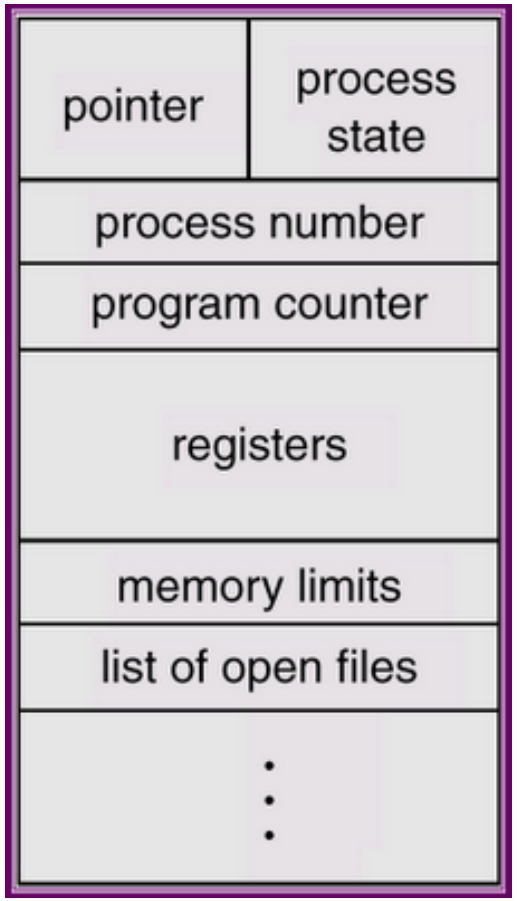


O que é Thread?

- Antes de definir threads vamos entender o que é um PROCESSO.
- Em um processo existe uma thread (uma linha de execução)

PROCESSO

- Um processo é uma instância de um programa que está em *execução*
- Está em um dos seus 5 estados
- Possui vários atributos no seu *Process Control Block*



O que é Thread?

- Antes de definir threads vamos entender o que é um PROCESSO.
- Em um processo existe uma thread (uma linha de execução)

Thread

Thread (linha de execução) é o menor fluxo sequencial de execução dentro de um programa, o qual pode ser gerenciado por um escalonador.

O que é Thread?

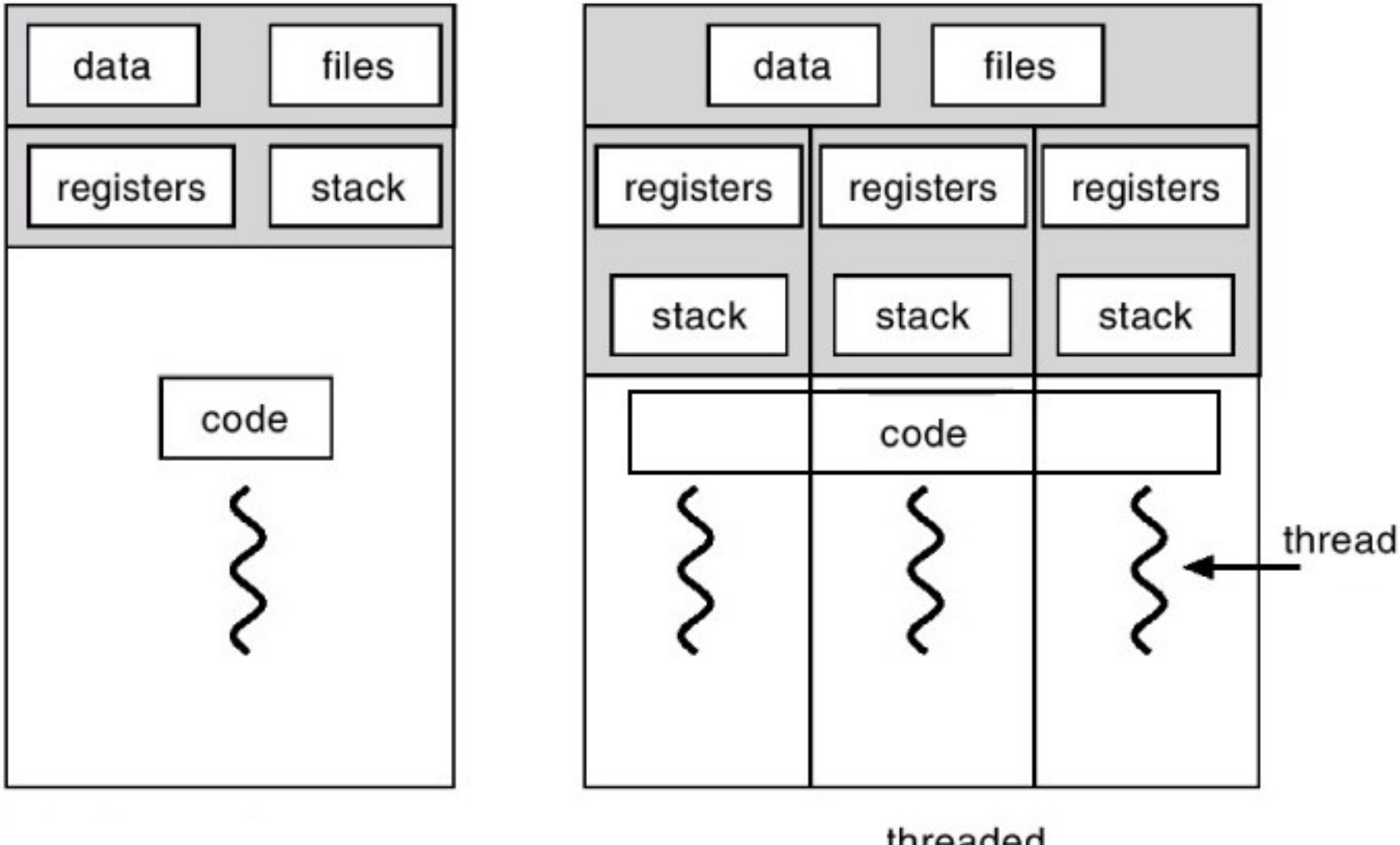
- Antes de definir threads vamos entender o que é um PROCESSO.
- Em um processo existe uma thread (uma linha de execução)

Thread

Thread (linha de execução) é o menor fluxo sequencial de execução dentro de um programa, o qual pode ser gerenciado por um escalonador.

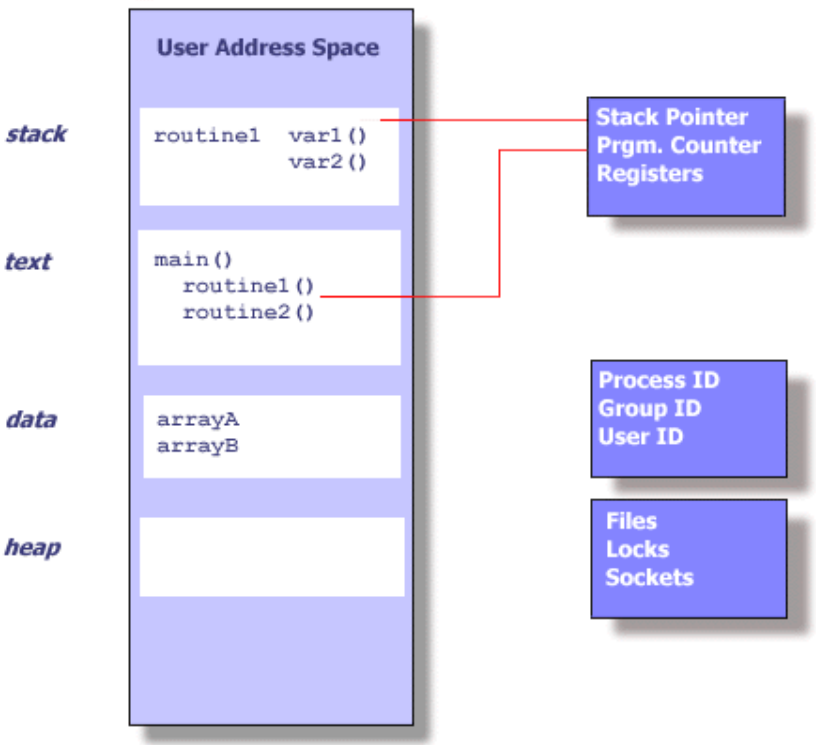
Thread

Thread (linha de execução) é o menor fluxo sequencial de execução dentro de um programa, o qual pode ser gerenciado por um escalonador.



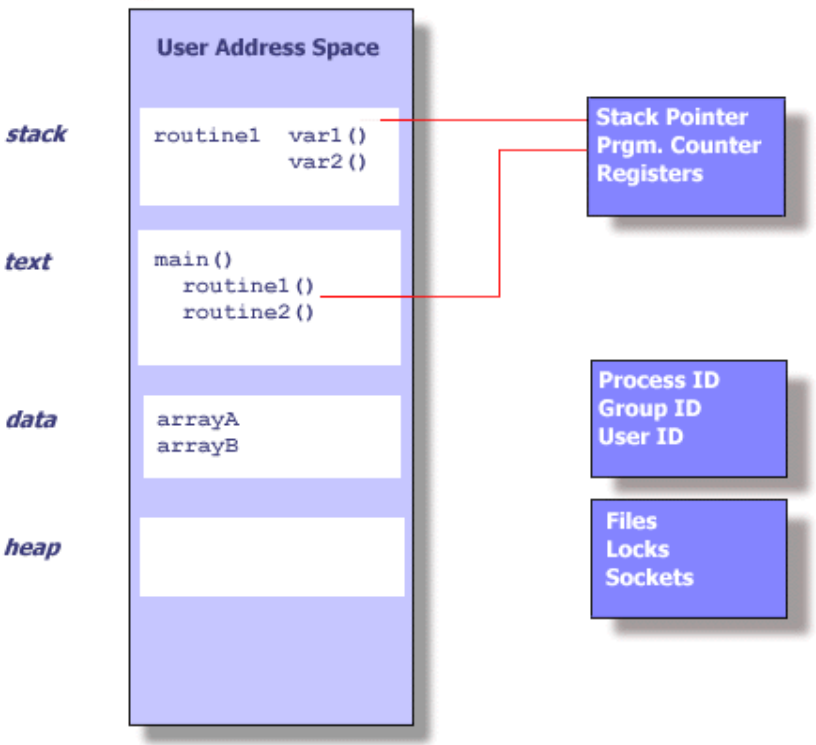
Thread

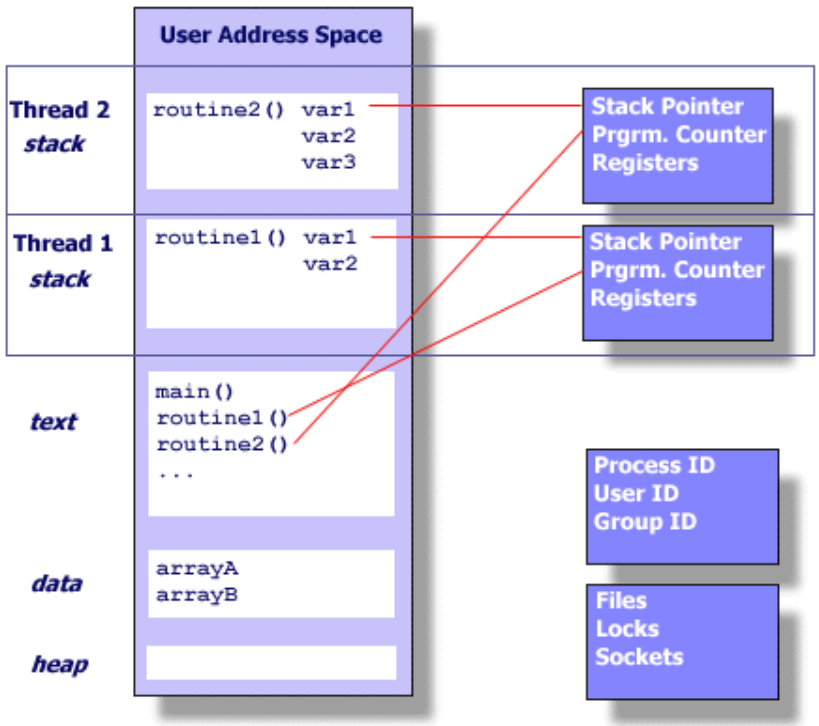
Thread (linha de execução) é o menor fluxo sequencial de execução dentro de um programa, o qual pode ser gerenciado por um escalonador.



Thread

Thread (linha de execução) é o menor fluxo sequencial de execução dentro de um programa, o qual pode ser gerenciado por um escalonador.





Diferenças entre Threads e Processos

Process	Thread
Process is considered heavy weight	Thread is considered light weight
Unit of Resource Allocation and of protection	Unit of CPU utilization
Process creation is very costly in terms of resources	Thread creation is very economical
Program executing as process are relatively slow	Programs executing using thread are comparatively faster
Process cannot access the memory area belonging to another process	Thread can access the memory area belonging to another thread within the same process
Process switching is time consuming	Thread switching is faster
One Process can contain several threads	One thread can belong to exactly one process

O que é PThreads?

Através da biblioteca Pthreads – Visão Geral

- Historicamente grandes empresas de HW tinham suas próprias implementações de threads
- Isto gerou:
- Então foi necessário criar uma padronização

Links Úteis

- https://computing.llnl.gov/tutorials/pthreads/
- http://www.opengroup.org/austin/papers/posix_faq.html

O que é PThreads?

Através da biblioteca Pthreads – Visão Geral

- Historicamente grandes empresas de HW tinham suas próprias implementações de threads
- Isto gerou:
  - Dificuldades de desenvolvimento e portabilidade
- Então foi necessário criar uma padronização

Links Úteis

- https://computing.llnl.gov/tutorials/pthreads/
- http://www.opengroup.org/austin/papers/posix_faq.html

O que é PThreads?

Através da biblioteca Pthreads – Visão Geral

- Historicamente grandes empresas de HW tinham suas próprias implementações de threads
- Isto gerou:
  - Dificuldades de desenvolvidmento e portabilidade
- Então foi necessário criar uma padronização

Links Úteis

- https://computing.llnl.gov/tutorials/pthreads/
- http://www.opengroup.org/austin/papers/posix_faq.html

O que é PThreads?

Através da biblioteca Pthreads – Visão Geral

- Historicamente grandes empresas de HW tinham suas próprias implementações de threads
- Isto gerou:
  - Dificuldades de desenvolvidmento e portabilidade
- Então foi necessário criar uma padronização
  - Para sistemas UNIX (Solares, Linux) foi especificado o padrão IEEE POSIX 1003.1c standard (1995)[a]
  - Implementações que usam esse padrão são referenciadas como POSIX threads (Pthreads)

[a] Existe implementações para Windows

Links Úteis

- https://computing.llnl.gov/tutorials/pthreads/
- http://www.opengroup.org/austin/papers/posix_faq.html

O que é PThreads?

Através da biblioteca Pthreads – Visão Geral

- Historicamente grandes empresas de HW tinham suas próprias implementações de threads
- Isto gerou:
  - Dificuldades de desenvolvidmento e portabilidade
- Então foi necessário criar uma padronização
  - Para sistemas UNIX (Solares, Linux) foi especificado o padrão IEEE POSIX 1003.1c standard (1995)[a]
  - Implementações que usam esse padrão são referenciadas como POSIX threads (Pthreads)

[a] Existe implementações para Windows

Links Úteis

- https://computing.llnl.gov/tutorials/pthreads/
- http://www.opengroup.org/austin/papers/posix_faq.html

O que é PThreads?

Através da biblioteca Pthreads – Visão Geral

- Historicamente grandes empresas de HW tinham suas próprias implementações de threads
- Isto gerou:
  - Dificuldades de desenvolvidmento e portabilidade
- Então foi necessário criar uma padronização
  - Para sistemas UNIX (Solares, Linux) foi especificado o padrão IEEE POSIX 1003.1c standard (1995)[a]
  - Implementações que usam esse padrão são referenciadas como POSIX threads (Pthreads)

[a] Existe implementações para Windows

Links Úteis

- https://computing.llnl.gov/tutorials/pthreads/
- http://www.opengroup.org/austin/papers/posix_faq.html

O que é PThreads?

Através da biblioteca Pthreads – Visão Geral

- Historicamente grandes empresas de HW tinham suas próprias implementações de threads
- Isto gerou:
  - Dificuldades de desenvolvimento e portabilidade
- Então foi necessário criar uma padronização
  - Para sistemas UNIX (Solares, Linux) foi especificado o padrão IEEE POSIX 1003.1c standard (1995)[a]
  - Implementações que usam esse padrão são referenciadas como POSIX threads (Pthreads)

[a] Existe implementações para Windows

Links Úteis

- `https://computing.llnl.gov/tutorials/pthreads/`
- `http://www.opengroup.org/austin/papers/posix_faq.html`

Pthread API

- As subrotinas da API Pthreads podem ser agrupadas em 4 grandes grupos:
  1. **Gerenciamento de Threads:** rotinas para criar, desanexar, agrupar. Além de funções para consultar atributos das threads
  2. **Mutexes:** rotinas para criar sincronizações para áreas críticas. Existem funções para criar, destruir, travar e destravar mutexes
  3. **Condition variables:** rotinas para lidar com a comunicação entre threads que compartilham um mutex
  4. **Sincronização:** rotinas que gerenciam leituras/escritas em seções críticas

Pthread API

- As subrotinas da API Pthreads podem ser agrupadas em 4 grandes grupos:
  1. **Gerenciamento de Threads:** rotinas para criar, desanexar, agrupar. Além de funções para consultar atributos das threads
  2. **Mutexes:** rotinas para criar sincronizações para áreas críticas. Existem funções para criar, destruir, travar e destravar mutexes
  3. **Condition variables:** rotinas para lidar com a comunicação entre threads que compartilham um mutex
  4. **Sincronização:** rotinas que gerenciam leituras/escritas em seções críticas

Pthread API

- As subrotinas da API Pthreads podem ser agrupadas em 4 grandes grupos:
  1. **Gerenciamento de Threads:** rotinas para criar, desanexar, agrupar. Além de funções para consultar atributos das threads
  2. **Mutexes:** rotinas para criar sincronizações para áreas críticas. Existem funções para criar, destruir, travar e destravar mutexes
  3. **Condition variables:** rotinas para lidar com a comunicação entre threads que compartilham um mutex
  4. **Sincronização:** rotinas que gerenciam leituras/escritas em seções críticas

Pthread API

- As subrotinas da API Pthreads podem ser agrupadas em 4 grandes grupos:
  1. **Gerenciamento de Threads:** rotinas para criar, desanexar, agrupar. Além de funções para consultar atributos das threads
  2. **Mutexes:** rotinas para criar sincronizações para áreas críticas. Existem funções para criar, destruir, travar e destravar mutexes
  3. **Condition variables:** rotinas para lidar com a comunicação entre threads que compartilham um mutex
  4. **Sincronização:** rotinas que gerenciam leituras/escritas em seções críticas

Pthread API

- As subrotinas da API Pthreads podem ser agrupadas em 4 grandes grupos:
  1. **Gerenciamento de Threads:** rotinas para criar, desanexar, agrupar. Além de funções para consultar atributos das threads
  2. **Mutexes:** rotinas para criar sincronizações para áreas críticas. Existem funções para criar, destruir, travar e destravar mutexes
  3. **Condition variables:** rotinas para lidar com a comunicação entre threads que compartilham um mutex
  4. **Sincronização:** rotinas que gerenciam leituras/escritas em seções críticas

```
int pthread_create(pthread_t * thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void * arg);
```

Criar uma thread

- pthread_t * thread: identificador único para uma nova thread (retornado pela subrotina)
- pthread_attr_t *attr: estrutura para definir valores dos atributos da thread
- void * (*start_routine)(void *): uma FUNÇÃO EM C QUE A THREAD EXECUTARÁ
- void * arg[a]: arumento recebido por "*start_routine". Use NULL para valores padrão.

[a] Esse argumento deve ser passado por referência. Atente-se para o cast para void

```
int pthread_create(pthread_t * thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void * arg);
```

Criar uma thread

- pthread_t * thread: identificador único para uma nova thread (retornado pela subrotina)
- pthread_attr_t *attr: estrutura para definir valores dos atributos da thread
  - Política de escalonamento
  - Tamanho da pilha...
- void * (*start_routine)(void *): uma FUNÇÃO EM C QUE A THREAD EXECUTARÁ
- void * arg[a]: arumento recebido por "*start_routine". Use NULL para valores padrão.

[a] Esse argumento deve ser passado por referência. Atente-se para o cast para void

```
int pthread_create(pthread_t * thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void * arg);
```

Criar uma thread

- pthread_t * thread: identificador único para uma nova thread (retornado pela subrotina)
- pthread_attr_t *attr: estrutura para definir valores dos atributos da thread
  - Política de escalonamento
  - Tamanho da pilha...
- void * (*start_routine)(void *): uma FUNÇÃO EM C QUE A THREAD EXECUTARÁ
- void * arg[a]: arumento recebido por "*start_routine". Use NULL para valores padrão.

[a] Esse argumento deve ser passado por referência. Atente-se para o cast para void

```
int pthread_create(pthread_t * thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void * arg);
```

Criar uma thread

- pthread_t * thread: identificador único para uma nova thread (retornado pela subrotina)
- pthread_attr_t *attr: estrutura para definir valores dos atributos da thread
  - Política de escalonamento
  - Tamanho da pilha...
- void * (*start_routine)(void *): uma FUNÇÃO EM C QUE A THREAD EXECUTARÁ
- void * arg[a]: arumento recebido por "*start_routine". Use NULL para valores padrão.

[a] Esse argumento deve ser passado por referência. Atente-se para o cast para void

```
int pthread_create(pthread_t * thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void * arg);
```

Criar uma thread

- pthread_t * thread: identificador único para uma nova thread (retornado pela subrotina)
- pthread_attr_t *attr: estrutura para definir valores dos atributos da thread
  - Política de escalonamento
  - Tamanho da pilha...
- void * (*start_routine)(void *): uma FUNÇÃO EM C QUE A THREAD EXECUTARÁ
- void * arg[a]: arumento recebido por "*start_routine". Use NULL para valores padrão.

[a] Esse argumento deve ser passado por referência. Atente-se para o cast para void

```
int pthread_create(pthread_t * thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void * arg);
```

Criar uma thread

- pthread_t * thread: identificador único para uma nova thread (retornado pela subrotina)
- pthread_attr_t *attr: estrutura para definir valores dos atributos da thread
  - Política de escalonamento
  - Tamanho da pilha...
- void * (*start_routine)(void *): uma FUNÇÃO EM C QUE A THREAD EXECUTARÁ
- void * arg[a]: arumento recebido por "*start_routine". Use NULL para valores padrão.

[a] Esse argumento deve ser passado por referência. Atente-se para o cast para void

```
int pthread_create(pthread_t * thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void * arg);
```

Retorno

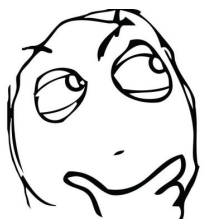
- 0, se tudo Ok!
- Valor indicando um dos possíveis erros.

Question time!

- Q: Dado que a thread foi criada como você saberá:
  - Quando a thread será escalonada para executar pelo SO?
  - Qual processaor/núcleo ela irá executar?

Resposta

Programas robustos não depedem de uma ordem específica para executar as threads.
O SO irá decidir **quando e onde** executar as treads.
NÃO PRECISAMOS TRATAR ISSO!



Question time!

- Q: Dado que a thread foi criada como você saberá:
  - Quando a thread será escalonada para executar pelo SO?
  - Qual processaor/núcleo ela irá executar?

Resposta

Programas robustos não depedem de uma ordem específica para executar as threads.
O SO irá decidir **quando e onde** executar as treads.
NÃO PRECISAMOS TRATAR ISSO!

void pthread_exit(void * retval);

Terminar a execução thread

- void * retval: valor de retorno da thread
  - Uma thread por terminar por vários motivos:
    - Retornam normalmente da sua função "*start_routine"
    - Fazendo uma chamada para a subrotina pthread_exit
    - Pode ser cancelada por outra thread via subrotina pthread_cancel
    - ...

Dica

- Se a thread mãe termina retornando da sua função principal, então suas filhas morrem.
- Se a thread mãe termina com pthread_exit(), suas filhas não morrem.

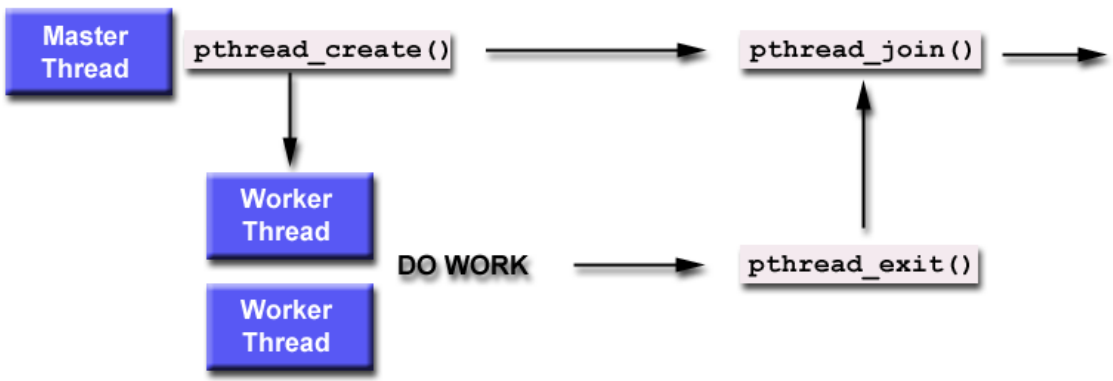
void pthread_join(pthread_t * thread, pthread_attr_t **status);

Terminar a execução thread

- pthread_t * thread: indica a thread a ser aguardada
- pthread_attr_t **status: valor de retorno da thread

Dica

Essa função faz com que a thread que a chamou espere até que a thread passada como parâmetro retorne



```c
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0; t<NUM_THREADS; t++){
      printf("In main: creating thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```

Compiler / Platform	Compiler Command
INTEL Linux	icc -pthread
	icpc -pthread
PGI Linux	pgcc -lpthread
	pgCC -lpthread
GNU Linux, Blue Gene	gcc -pthread
	g++ -pthread
IBM Blue Gene	bgxlc_r / bgcc_r
	bgxlC_r, bgxlc++_r

```c
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS    5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0; t<NUM_THREADS; t++){
      printf("In main: creating thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```

Saída

```
In main:  creating thread 0
In main:  creating thread 1
Hello World!  It's me, thread #0!
In main:  creating thread 2
Hello World!  It's me, thread #1!
Hello World!  It's me, thread #2!
In main:  creating thread 3
In main:  creating thread 4
Hello World!  It's me, thread #4!
Hello World!  It's me, thread #3!
```

Figura : Vlw Brunão ta Serto, mals i daew?
R: RLX!

Como uso thread com sockets?

- Relembrando, nosso OBJETIVO ERA:
  - Saber THREADS (✓)
  - Conexões simultaneas com Threads ( )

O que muda ao usar Threads?

```c
#include<stdio.h>
#include<string.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write

int main(int argc , char *argv[])
{
    int socket_desc , client_sock , c , read_size;
    struct sockaddr_in server , client;
    char client_message[2000];

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
```

```c
#include<stdio.h>
#include<string.h>    //strlen
#include<stdlib.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write
#include<pthread.h> //for threading , link with lpthread

//the thread function
void *connection_handler(void *);

int main(int argc , char *argv[])
{
    int socket_desc , client_sock , c , *new_sock;
    struct sockaddr_in server , client;
```

O que muda ao usar Threads?

```
 9  #include<stdio.h>
10  #include<string.h>    //strlen
11  #include<sys/socket.h>
12  #include<arpa/inet.h> //inet_addr
13  #include<unistd.h>    //write
14
15  int main(int argc , char *argv[])
16  {
17      int socket_desc , client_sock , c , read_size;
18      struct sockaddr_in server , client;
19      char client_message[2000];
20
21      //Create socket
22      socket_desc = socket(AF_INET , SOCK_STREAM , 0);
23      if (socket_desc == -1)
```

```
 9  #include<stdio.h>
10  #include<string.h>    //strlen
11  #include<stdlib.h>    //strlen
12  #include<sys/socket.h>
13  #include<arpa/inet.h> //inet_addr
14  #include<unistd.h>    //write
15  #include<pthread.h> //for threading , link with lpthread
16
17  //the thread function
18  void *connection_handler(void *);
19
20  int main(int argc , char *argv[])
21  {
22      int socket_desc , client_sock , c , *new_sock;
23      struct sockaddr_in server , client;
```

O que muda ao usar Threads?

Processo com 1 Thread (sem conexões simultaneas)

```
50      //accept connection from an incoming client
51      client_sock = accept(socket_desc,
52                           (struct sockaddr *)&client,
53                           (socklen_t*)&c);
54      if (client_sock < 0)
55      {
56          perror("accept failed");
57          return 1;
58      }
59      puts("Connection accepted");
60
61      //Receive a message from client
62      while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
63      {
64          //Send the message back to client
65          write(client_sock , client_message , strlen(client_message));
66      }
```

O que muda ao usar Threads?

Com PThread (permite-se conexões simultaneas)

```
55      //Accept and incoming connection
56      puts("Waiting for incoming connections...");
57      c = sizeof(struct sockaddr_in);
58
59      while( (client_sock = accept(socket_desc,
60                                   (struct sockaddr *)&client,
61                                   (socklen_t*)&c)) )
62      {
63          puts("Connection accepted");
64
65          pthread_t sniffer_thread;
66          new_sock = malloc(1);
67          *new_sock = client_sock;
68
69          if( pthread_create( &sniffer_thread ,
70                             NULL ,
71                             connection_handler ,
72                             (void*) new_sock) < 0)
73          {
74              perror("could not create thread");
75              return 1;
76          }
77
78          //Now join the thread , so that we dont terminate before the thread
79          //pthread_join( sniffer_thread , NULL);
80          puts("Handler assigned");
81      }
```

O que muda ao usar Threads?

Com PThread (permite-se conexões simultaneas)

```
55      //Accept and incoming connection
56      puts("Waiting for incoming connections...");
57      c = sizeof(struct sockaddr_in);
58
59      while( (client_sock = accept(socket_desc,
60                                   (struct sockaddr *)&client,
61                                   (socklen_t*)&c)) )
62      {
63          puts("Connection accepted");
64
65          pthread_t sniffer_thread;
66          new_sock = malloc(1);
67          *new_sock = client_sock;
68
69          if( pthread_create( &sniffer_thread ,
70                              NULL ,
71                              connection_handler ,
72                              (void*) new_sock) < 0)
73          {
74              perror("could not create thread");
75              return 1;
76          }
77
78          //Now join the thread , so that we dont terminate before the thread
79          //pthread_join( sniffer_thread , NULL);
80          puts("Handler assigned");
81      }
```

O que muda ao usar Threads?

Com PThread (permite-se conexões simultaneas)

```
55      //Accept and incoming connection
56      puts("Waiting for incoming connections...");
57      c = sizeof(struct sockaddr_in);
58
59      while( (client_sock = accept(socket_desc,
60                                   (struct sockaddr *)&client,
61                                   (socklen_t*)&c)) )
62      {
63          puts("Connection accepted");
64
65          pthread_t sniffer_thread;
66          new_sock = malloc(1);
67          *new_sock = client_sock;
68
69          if( pthread_create( &sniffer_thread ,
70                              NULL ,
71                              connection_handler ,
72                              (void*) new_sock) < 0)
73          {
74              perror("could not create thread");
75              return 1;
76          }
77
78          //Now join the thread , so that we dont terminate before the thread
79          //pthread_join( sniffer_thread , NULL);
80          puts("Handler assigned");
81      }
```

O que muda ao usar Threads?

Com PThread (permite-se conexões simultaneas)

```
55      //Accept and incoming connection
56      puts("Waiting for incoming connections...");
57      c = sizeof(struct sockaddr_in);
58
59      while( (client_sock = accept(socket_desc,
60                                   (struct sockaddr *)&client,
61                                   (socklen_t*)&c)) )
62      {
63          puts("Connection accepted");
64
65          pthread_t sniffer_thread;
66          new_sock = malloc(1);
67          *new_sock = client_sock;
68
69          if( pthread_create( &sniffer_thread ,
70                              NULL ,
71                              connection_handler ,
72                              (void*) new_sock) < 0)
73          {
74              perror("could not create thread");
75              return 1;
76          }
77
78          //Now join the thread , so that we dont terminate before the thread
79          //pthread_join( sniffer_thread , NULL);
80          puts("Handler assigned");
81      }
```

O que muda ao usar Threads?

Com PThread (permite-se conexões simultaneas)

connection-handle

```
 95 void *connection_handler(void *socket_desc) {
 96     //Get the socket descriptor
 97     int sock = *(int*)socket_desc;
 98     int read_size;
 99     char *message , client_message[2000];
100
101     message = "Now type something and i shall repeat what you type \n";
102     write(sock , message , strlen(message));
103
104     //Receive a message from client
105     while( (read_size = recv(sock , client_message , 2000 , 0)) > 0 ){
106         //Send the message back to client
107         write(sock , client_message , strlen(client_message));
108     }
109
110     if(read_size == 0){
111         puts("Client disconnected");
112         fflush(stdout);
113     }
114     else if(read_size == -1) {
115         perror("recv failed");
116     }
117
118     //Free the socket pointer
119     free(socket_desc);
120
121     return 0;
122 }
```

O que muda ao usar Threads?

Sem PThread

```
50    //accept connection from an incoming client
51    client_sock = accept(socket_desc,
52                         (struct sockaddr *)&client,
53                         (socklen_t*)&c);
54    if (client_sock < 0)
55    {
56        perror("accept failed");
57        return 1;
58    }
59    puts("Connection accepted");
60
61    //Receive a message from client
62    while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
63    {
64        //Send the message back to client
65        write(client_sock , client_message , strlen(client_message));
66    }
```

Com PThread

```
55    //Accept and incoming connection
56    puts("Waiting for incoming connections...");
57    c = sizeof(struct sockaddr_in);
58
59    while( (client_sock = accept(socket_desc,
60                                 (struct sockaddr *)&client,
61                                 (socklen_t*)&c)) )
62    {
63        puts("Connection accepted");
64
65        pthread_t sniffer_thread;
66        new_sock = malloc(1);
67        *new_sock = client_sock;
68
69        if( pthread_create( &sniffer_thread ,
70                            NULL ,
71                            connection_handler ,
72                            (void*) new_sock) < 0)
73        {
74            perror("could not create thread");
75            return 1;
76        }
77
78        //Now join the thread , so that we dont terminate before the thread
79        //pthread_join( sniffer_thread , NULL);
80        puts("Handler assigned");
81    }
```

Executar códigos

Parte II – Função select()

Objetivos

- Ao final dessa aula você será capaz de:
  - Entender o uso de threads
  - Entender o uso do *Select*
- Criar um programa **SERVIDOR** que aceita conexão de vários clientes
  - Usando *Threads*
  - Usando *Select*
  - Usanto libevent
- Vamos criar um **SERVIDOR** que faz uso da SELECT para gerenciar múltiplas conexões

Parte II – Função select()

Objetivos

- Ao final dessa aula você será capaz de:
  - Entender o uso de threads
  - Entender o uso do *Select*
- Criar um programa **SERVIDOR** que aceita conexão de vários clientes
  - Usando *Threads*
  - Usando *Select*
  - Usanto libevent
- Vamos criar um **SERVIDOR** que faz uso da SELECT para gerenciar múltiplas conexões

select() – Visão geral

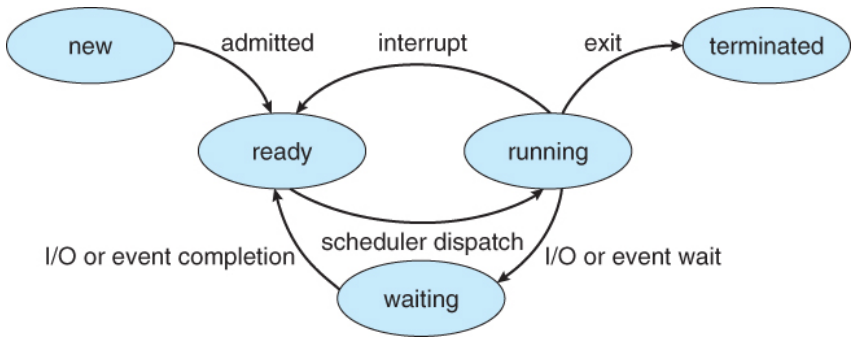
- Ao invés de um thread para cada requisição, um único processo[a] serve a todas as requisições
- select() funciona bloqueando o processo até que seja acordado por um evento no socket

[a] Com uma única thread

select() – Visão geral

- Ao invés de um thread para cada requisição, um único processo[a] serve a todas as requisições
- select() funciona bloqueando o processo até que seja acordado por um evento no socket

[a] Com uma única thread



select() – Vantagens e Desvantagens

Vantagens

- O servidor precisa de apenas um único processo para lidar com todas as solicitações
- O servidor não vai precisar de primitivas de memória compartilhada ou sincronização para que diferentes 'tarefas' se comuniquem

Desvantagens

- O servidor não pode agir como se houvesse apenas um cliente
- Com select(), a programação não é tão transparente

select() – Vantagens e Desvantagens

Vantagens

- O servidor precisa de apenas um único processo para lidar com todas as solicitações
- O servidor não vai precisar de primitivas de memória compartilhada ou sincronização para que diferentes 'tarefas' se comuniquem

Desvantagens

- O servidor não pode agir como se houvesse apenas um cliente
- Com select(), a programação não é tão transparente

Como o select() funciona?

- Select() funciona bloqueando o processo até que **algo** aconteça em um descritor de arquivo (ou seja, em um socket)
- O que é **algo**?
  - **Você diz** ao select() o que quer que esse **algo** que vai te acordar seja (Ex. entrada de dados)

Como o select() funciona?

- Select() funciona bloqueando o processo até que **algo** aconteça em um descritor de arquivo (ou seja, em um socket)
- O que é **algo**?
  - **Você diz** ao select() o que quer que esse **algo** que vai te acordar seja (Ex. entrada de dados)
  - Como assim "Você diz"?
    - Você usa a estrutura **fd_set** e algumas macros para isso.

Como o select() funciona?

- Select() funciona bloqueando o processo até que **algo** aconteça em um descritor de arquivo (ou seja, em um socket)
- O que é **algo**?
  - **Você diz** ao select() o que quer que esse **algo** que vai te acordar seja (Ex. entrada de dados)
  - Como assim "**Você diz**"?
    - Você usa a estrutura **fd_set** e algumas macros para isso.

fd_set

- **fd_set** é um conjunto de sockets para "monitorar" alguma atividade
- Há quadro macros úteis para manipular uma **fd_set**:

fd_set

- **fd_set** é um conjunto de sockets para "monitorar" alguma atividade
- Há quadro macros úteis para manipular uma **fd_set**:
  - FD_SET(int fd, fd_set *set); adiciona um fd ao set
  - FD_CLR(int fd, fd_set *set); remove fd se estiver no set
  - FD_ISSET(int fd, fd_set *set); retorna true se fd estiver no set
  - FD_ZERO(fd_set *set); remove todas as entradas do set

fd_set

- **fd_set** é um conjunto de sockets para "monitorar" alguma atividade
- Há quadro macros úteis para manipular uma **fd_set**:
  - FD_SET(int fd, fd_set *set); adiciona um fd ao set
  - FD_CLR(int fd, fd_set *set); remove fd se estiver no set
  - FD_ISSET(int fd, fd_set *set); retorna true se fd estiver no set
  - FD_ZERO(fd_set *set); remove todas as entradas do set

fd_set

- **fd_set** é um conjunto de sockets para "monitorar" alguma atividade
- Há quadro macros úteis para manipular uma **fd_set**:
  - FD_SET(int fd, fd_set *set); adiciona um fd ao set
  - FD_CLR(int fd, fd_set *set); remove fd se estiver no set
  - FD_ISSET(int fd, fd_set *set); retorna true se fd estiver no set
  - FD_ZERO(fd_set *set); remove todas as entradas do set

fd_set

- **fd_set** é um conjunto de sockets para "monitorar" alguma atividade
- Há quadro macros úteis para manipular uma **fd_set**:
  - FD_SET(int fd, fd_set *set); adiciona um fd ao set
  - FD_CLR(int fd, fd_set *set); remove fd se estiver no set
  - FD_ISSET(int fd, fd_set *set); retorna true se fd estiver no set
  - FD_ZERO(fd_set *set); remove todas as entradas do set

fd_set

- **fd_set** é um conjunto de sockets para "monitorar" alguma atividade
- Há quadro macros úteis para manipular uma **fd_set**:
  - FD_SET(int fd, fd_set *set); adiciona um fd ao set
  - FD_CLR(int fd, fd_set *set); remove fd se estiver no set
  - FD_ISSET(int fd, fd_set *set); retorna true se fd estiver no set
  - FD_ZERO(fd_set *set); remove todas as entradas do set

Usando o fd_set

- Cria-se uma estrutura para armazenar os sockets
  fd_set readfds;
- Adiciona-se um socket descriptor (sd) á estrutura
  FD_SET(sd, readfds);

Usando o fd_set

- Cria-se uma estrutura para armazenar os sockets
  fd_set readfds;
- Adiciona-se um socket descriptor (sd) á estrutura
  FD_SET(sd, readfds);

```
int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds,
struct timeval *timeout );
```

Função select()

- int nfds: tamanho da estrutura (limite superior)
- fd_set *read-fds: se você quer saber se algum socket está pronto para receber
- fd_set *write-fds: se você quer saber se algum socket está pronto para enviar
- fd_set *except-fds: se você quer saber se alguma exceção ocorreu em algum socket
- struct timeval *timeout: por quanto tempo verificar esses conjuntos para você

Info.

select() nos dá o "poder"para monitorar vários sockets ao mesmo tempo. Ela informará qual socket está pronto para ler/escrever ou qual lançou exceções.

int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds, struct timeval *timeout);

Função select()

- int nfds: tamanho da estrutura (limite superior)
- fd_set *read-fds: se você quer saber se algum socket está pronto para receber
- fd_set *write-fds: se você quer saber se algum socket está pronto para enviar
- fd_set *except-fds: se você quer saber se alguma exceção ocorreu em algum socket
- struct timeval *timeout: por quanto tempo verificar esses conjuntos para você

Info.

select() nos dá o "poder"para monitorar vários sockets ao mesmo tempo. Ela informará qual socket está pronto para ler/escrever ou qual lançou exceções.

int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds, struct timeval *timeout);

Função select()

- int nfds: tamanho da estrutura (limite superior)
- fd_set *read-fds: se você quer saber se algum socket está pronto para receber
- fd_set *write-fds: se você quer saber se algum socket está pronto para enviar
- fd_set *except-fds: se você quer saber se alguma exceção ocorreu em algum socket
- struct timeval *timeout: por quanto tempo verificar esses conjuntos para você

Info.

select() nos dá o "poder" para monitorar vários sockets ao mesmo tempo. Ela informará qual socket está pronto para ler/escrever ou qual lançou exceções.

int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds, struct timeval *timeout);

Função select()

- int nfds: tamanho da estrutura (limite superior)
- fd_set *read-fds: se você quer saber se algum socket está pronto para receber
- fd_set *write-fds: se você quer saber se algum socket está pronto para enviar
- fd_set *except-fds: se você quer saber se alguma exceção ocorreu em algum socket
- struct timeval *timeout: por quanto tempo verificar esses conjuntos para você

Info.

select() nos dá o "poder" para monitorar vários sockets ao mesmo tempo. Ela informará qual socket está pronto para ler/escrever ou qual lançou exceções.

int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds, struct timeval *timeout);

Função select()

- int nfds: tamanho da estrutura (limite superior)
- fd_set *read-fds: se você quer saber se algum socket está pronto para receber
- fd_set *write-fds: se você quer saber se algum socket está pronto para enviar
- fd_set *except-fds: se você quer saber se alguma exceção ocorreu em algum socket
- struct timeval *timeout: por quanto tempo verificar esses conjuntos para você

Info.

select() nos dá o "poder" para monitorar vários sockets ao mesmo tempo. Ela informará qual socket está pronto para ler/escrever ou qual lançou exceções.

int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds, struct timeval *timeout);

Função select()

- int nfds: tamanho da estrutura (limite superior)
- fd_set *read-fds: se você quer saber se algum socket está pronto para receber
- fd_set *write-fds: se você quer saber se algum socket está pronto para enviar
- fd_set *except-fds: se você quer saber se alguma exceção ocorreu em algum socket
- struct timeval *timeout: por quanto tempo verificar esses conjuntos para você

Info.

select() nos dá o "poder" para monitorar vários sockets ao mesmo tempo. Ela informará qual socket está pronto para ler/escrever ou qual lançou exceções.

```
int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds,
struct timeval *timeout );
```

Função select()

- A função recebe uma lista de sockets para monitorar

```
int atividade;
atividade = select(max_fd + 1, &readfds, NULL, NULL, NULL );
```

- Quando um socket estiver pronto para ser lido, select vai retornar e readfds terá esses sockets que estão pronto para serem lidos

Retorno da função select()

- # de Descritores em caso de sucesso
- 0 se o timeout foi alcançado
- -1 em caso de erro

```
int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds,
struct timeval *timeout );
```

Função select()

- A função recebe uma lista de sockets para monitorar
  ```
  int atividade;
  atividade = select(max_fd + 1, &readfds, NULL, NULL, NULL );
  ```
- Quando um socket estiver pronto para ser lido, select vai retornar e readfds terá esses sockets que estão pronto para serem lidos

Retorno da função select()

- # de Descritores em caso de sucesso
- 0 se o timeout foi alcançado
- -1 em caso de erro

```
int select(int nfds, fd_set *read-fds, fd_set *write-fds, fd_set *except-fds,
struct timeval *timeout );
```

Função select()

- A função recebe uma lista de sockets para monitorar
  ```
  int atividade;
  atividade = select(max_fd + 1, &readfds, NULL, NULL, NULL );
  ```
- Quando um socket estiver pronto para ser lido, select vai retornar e readfds terá esses sockets que estão pronto para serem lidos

Retorno da função select()

- # de Descritores em caso de sucesso
- 0 se o timeout foi alcançado
- -1 em caso de erro

Sequência frequentemente usada

1. Preencha uma estrutura fd_set com o(s) socket(s) que, quando receber(em) dados, você quer ser informado
2. Preencha uma estrutura fd_set com o(s) socket(s) que, quando você puder escrever nele(s), você quer ser informado
3. Chame select() e bloqueie até que algo ocorra
4. Quando o select() retornar, confira se algum socket está pronto. Em caso afirmativo, sirva o socket da maneira necessária

Sequência frequentemente usada

1. Preencha uma estrutura fd_set com o(s) socket(s) que, quando receber(em) dados, você quer ser informado
2. Preencha uma estrutura fd_set com o(s) socket(s) que, quando você puder escrever nele(s), você quer ser informado
3. Chame select() e bloqueie até que algo ocorra
4. Quando o select() retornar, confira se algum socket está pronto. Em caso afirmativo, sirva o socket da maneira necessária

Sequência frequentemente usada

1. Preencha uma estrutura fd_set com o(s) socket(s) que, quando receber(em) dados, você quer ser informado
2. Preencha uma estrutura fd_set com o(s) socket(s) que, quando você puder escrever nele(s), você quer ser informado
3. Chame select() e bloqueie até que algo ocorra
4. Quando o select() retornar, confira se algum socket está pronto. Em caso afirmativo, sirva o socket da maneira necessária

Sequência frequentemente usada

1. Preencha uma estrutura fd_set com o(s) socket(s) que, quando receber(em) dados, você quer ser informado
2. Preencha uma estrutura fd_set com o(s) socket(s) que, quando você puder escrever nele(s), você quer ser informado
3. Chame select() e bloqueie até que algo ocorra
4. Quando o select() retornar, confira se algum socket está pronto. Em caso afirmativo, sirva o socket da maneira necessária

Figura : Vlw Brunão ta Serto, mals i daew?
R: RLX!

Como uso select com sockets?

- Relembrando, nosso OBJETIVO ERA:
  - Saber usar SELECT (✓)
  - Conexões simultaneas SELECT ( )

Exibir e executar códigos

Extras

MONITORIA

Seg e Qua 16hrs às 18 hrs

E-mail: **bruno.ps@dcc.ufmg.br**

Enviar e-mail antecipadamente com o dia/horário para o encontro com:

- Assunto: [MONITORIA] [ASSUNTO] [DIA/HORÁRIO]
  - ASSUNTO: TP, prova, lista de exercício ...
- Corpo: Ideia geral do problema.

Git é vida!

git clone https://github.com/BrunoPereiraSantos/aula-threads.git

Brian "Beej Jorgensen" Hall.
Beej's Guide to Network Programming.
http://beej.us/guide/bgnet/output/html/multipage/index.html.

Bruno Pereira.
Apresentação.
UFMG, 2014.
https://copy.com/94BD03mWisUf.

Bruno Pereira.
Códigos das aplicações apresentadas e extras.
UFMG, 2014.
https://copy.com/r1XTNWP4H3aM.

Larry L Peterson and Bruce S Davie.
Computer networks: a systems approach.
Elsevier, 2007.

Jon Postel.
User datagram protocol.
Isi, 1980.
http://tools.ietf.org/html/rfc768.

W Richard Stevens.
Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms.
1997.
http://tools.ietf.org/html/rfc2001.