Sistemas Operativos: Threads

Pedro F. Souto (pfs@fe.up.pt)

March 23, 2015

Roadmap

What is a Thread?

Use of Threads

Libpthreads

Multithreaded Programming Challenges

Further Reading

Interprocess Communication in Unix

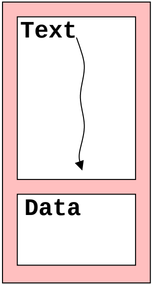
- In Unix-like OSs, e.g. Linux, a process runs in a virtual processor:
  - each process has the impression that it has all computer resources at its disposal
- Communication between processes in Unix is not easy:
  - the parent process can pass whatever information it wishes to its child process upon its creation, but afterwards ...
  - a child process can pass a very limited amount of information only to its parent upon its termination
  - sinchronization among processes is possible only between a parent process and its children
- More recently, Unix-like OS also support shared memory among processes:
  - + makes it easy for processes to cooperate;
  - – its use is not very convenient ;
  - – it is relatively inefficient as processes must synchronize via the OS

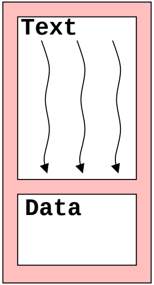
Threads

Threads abstract the execution of a sequence of instructions, i.e. a thread of execution

Simplifying, whereas a process abstracts the execution of a program, a thread abstracts the execution of a function

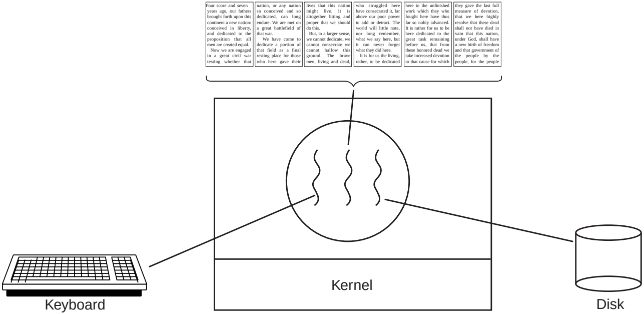
- In more recent OSs, a process may provide an execution environment for more than one thread.





Multithreaded Text Processor

The idea is to use one thread per task



Four score and seven years ago, our fathers brought forth upon this continent a new nation: conceived in liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield of that war.

We have come to dedicate a portion of that field as a final resting place for those who here gave their lives that this nation might live. It is altogether fitting and proper that we should do this.

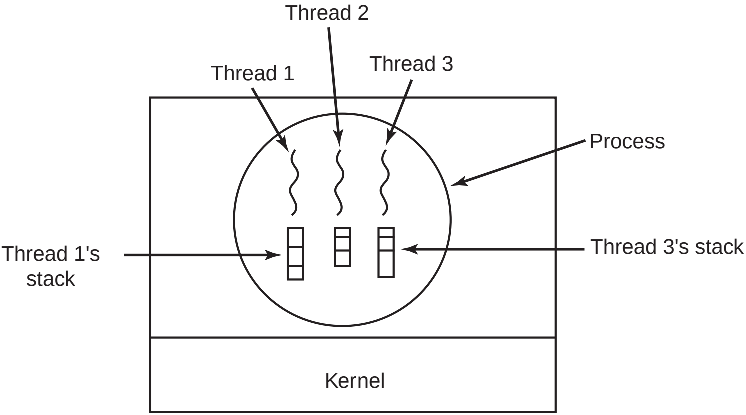
But, in a larger sense, we cannot dedicate, we cannot consecrate we cannot hallow this ground. The brave men, living and dead, who struggled here have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember, what we say here, but it can never forget what they did here.

It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us, that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion, that we here highly resolve that these dead shall not have died in vain that this nation, under God, shall have a new birth of freedom and that government of the people by the people, for the people

1. One thread interfaces with the user (via the keyboard, the mouse and the screen);
2. One thread formats the text in backgroung
3. One thread periodically saves the file on non-volatile storage, e.g. hard disk.

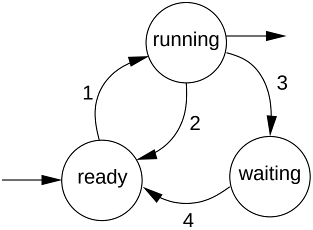
Resource Sharing with Threads

- Threads of a given process may share most resources, except the stack and the processor state:



Thread State

- Like a process, a *thread* may be in one of 3 states:



- Thread-specific information is relatively small:
  - its state (e.g. a process may be blocked waiting for an event
  - the processo state (incluing the `SP` and `PC`);
  - a stack.
- Operations like:
  - creation/termination
  - switching

  on threads of the same process are much more efficient than the same operations on processes

Roadmap

What is a Thread?

Use of Threads

Libpthreads

Multithreaded Programming Challenges

Further Reading

Use of Threads

- Same process threads may share many resources, including the address space

  they are particularly appropriate for applications that comprise several **concurrent** activities

- E.g. Web server:
  - Receives and processes requests for Web pages.
  - Web pages are files stored on disk.
  - Keeps in main memory a cache of the pages most recently accessed
  - If the requested page is not in the cache, the server must go to disk

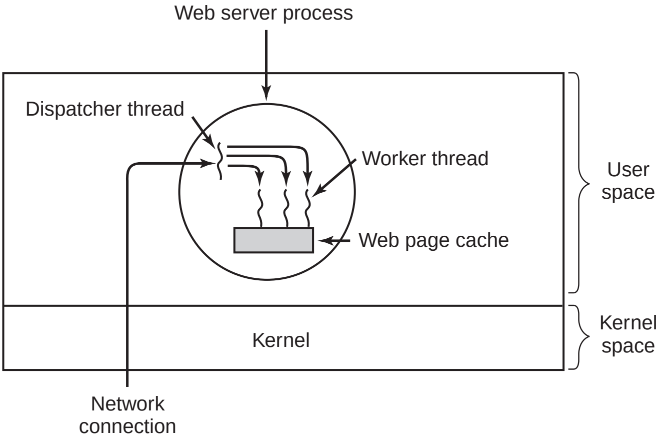
Singe Threaded Web Server

```
while( TRUE ) {
  get_next_request(&buf);
  lookup_page_in_cache(buf, &page);
  if( page == NULL )
     read_page_from_disk(buf, &page);
  send_page(page);
}
```

- If the page is not in the cache, the server must go to disk, blocking
- While the page is not brought to main memory, the server cannot process other requests
- The number that such server can process per time unit is rather low

Multi-Threaded Web Server

- A thread, the dispatcher, receives Web requests and passes them to worker threads
- Each worker thread processes one request at a time: no problem if it blocks on an I/O operation



Multi-Threaded Web Server (Code)

- Dispatcher thread:

```
while( TRUE ) {
    get_next_request(&buf);
    handoff_work(buf);
}
```

- Worker threads:

```
while( TRUE ) {
    wait_for_work(&buf);
    lookup_page_in_cache(buf, &page);
    if( page == NULL )
        read_page_from_disk(buf, &page);
    send_page(page);
}
```

Web Server Comparison

Architecture	**Parallelism**	**Ease of Programming**
Single threaded	No	Easy.
Multithreaded	Yes	May be hard.

Roadmap

What is a Thread?

Use of Threads

Libpthreads

Multithreaded Programming Challenges

Further Reading

libpthread (pthreads)

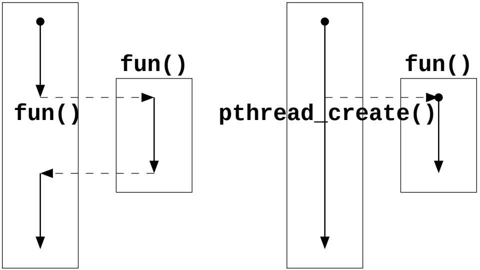
- POSIX thread library

  Specified to promote code portability

Life-cycle related pthread functions

`int pthread_create(pthread_t *id, ...)` creates a thread that executes the function specified in one of its arguments:

Function invocation vs thread creation



`void pthread_exit(void *value_ptr)` terminates the thread;

`int pthread_join(pthread_t thread, void **value_ptr`
waits for the termination of the thread specified in its first argument

Multithreaded Program Execution

- In a multithreaded program, a *thread* is created upon:
  - A program's creation: `main()` is executed by the main thread.
  - Execution of `pthread_create()`: all other threads
- A *thread* terminates if, e.g.:
  - it returns from the first function that it executed (`main()` pr `pthread_create()` argument;
  - it executes `pthread_exit()`.
- A multithreaded program terminates if, e.g.:
  - The main thread (see above) terminates;
  - Any *thread* invokes the `_exit()` system call

pthread_create()

```
int pthread_create(pthread_t *id,
                   const pthread_att_t attr,
                   void *(*start_fn)(void *), void *arg)
```

where:

`*id` is initialized inside by `pthread_create()` with the identity of the created *thread*;

`*attr` is a data structure that determines the attributes of the thread to be created (if `NULL` the thread will have default attributes)

`*start_fn` is the function the thread will execute. Its prototype is:

```
void *thr_fun(void *)
```

`*arg` is the argument passed to `thr_fun()`

pthread_create(): example

```
#include <pthread.h>
void *fun(void *arg) { /* Actually the ar- */
   ...                 /* gument is not used */
}
   ...
   pthread_attr_t attr;
   pthread_t      tid;
   ...
   pthread_attr_init(&attr); /* Initialize attr
                              * default values *
   pthread_create(&tid, &attr, fun, NULL);
   ...
```

- pthread_attr_init() initializes its argument to default values
- In general, the last argument of pthread_create() is the address of a data structure with the data to pass the function fun().

void *fun(void *arg)

Allows to define any function

```
#include <pthread.h>
   void *fun(void *arg) {
       args_t *my_args = args;
       ret_t *ret = malloc(sizeof(ret_t));
       ...
       return ret;
   }
```

Múltiple *Threads*

Normally multithreaded applications use more than 2 *threads*

- You need to allocate different variables for each thread

```
#include <pthread.h>
#define T 3 /* number of threads */
typedef void *(thr_fun_t)(void *arg);
   ...
   pthread_attr_t attr[T];
   pthread_t      tid[T];
   int            thr_arg[T];
   ...
   for( i = 0; i < T; i++ ) {
       pthread_attr_init(&attr); /* Initialize attr */
       pthread_create(&(tid[i]), &(attr[i]),
                      (thr_fun_t *) fun,
                      (void *)&(thr_arg[i]));
   }
   ...
```

Roadmap

What is a Thread?

Use of Threads

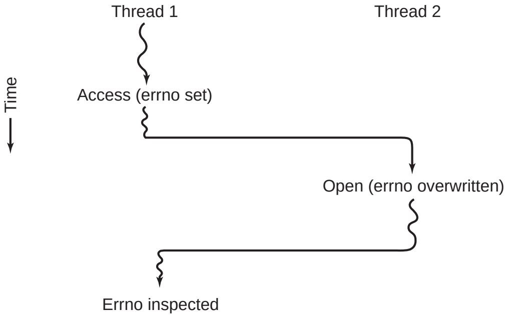
Libpthreads

Multithreaded Programming Challenges

Further Reading

Multithreaded Programming

- Legacy code written for single-threaded processes rarely works without changes in a multithreaded application:
  - global variables:



  - non-reentrant functions;
  - concurrency (*race conditions*).
- This is also true for libary code, including the C standard library:

  With `gcc`, you must use the `-pthread` option

Roadmap

What is a Thread?

Use of Threads

Libpthreads

Multithreaded Programming Challenges

Further Reading

Further Reading

OSTEP

- Ch. 36 (until Sec. 36.2): *Concurrency an Introduction*
- Ch. 27 (until Sec. 27.3): *Thread-API*

Sistemas Operativos

- Secção 3.4: *Modelo Multitarefa*
- Secção 3.6.4: *Tarefas - Interface POSIX*

Modern Operating Systems, 2nd. Ed.

- Section 2.2: *Threads*
- Section 2.2.8: *Making Single-Threaded Code Multithreaded*

Operating Systems Concepts

- Section 4.1: *Overview* (of threads)
- Section 4.3: *Thread Libraries* (only 4.3.1)
- Section 4.4: *Threading Issues* (for your education)