Thread Libraries: POSIX threads (pthreads)

Prof. Michele Loreti

Laboratorio di Sistemi Operativi

Corso di Laurea in Informatica (L31)

Scuola di Scienze e Tecnologie

Thread libraries...

There are two main thread libraries:

- POSIX threads, pthreads;
- Solaris threads, sthreads.

Thread libraries...

There are two main thread libraries:

- POSIX threads, pthreads;
- Solaris threads, sthreads.

Both contain code for:

- creating and destroying threads
- passing messages and data between threads
- scheduling thread execution
- saving and restoring thread contexts

Thread libraries...

There are two main thread libraries:

- POSIX threads, pthreads;
- Solaris threads, sthreads.

Both contain code for:

- creating and destroying threads
- passing messages and data between threads
- scheduling thread execution
- saving and restoring thread contexts

POSIX Threads Library

Creating a new thread...

A new thread is created via the function `pthread_create()`:

```
#include <pthread.h>

int pthread_create(
  pthread_t *tid,
  const pthread_attr_t *tattr,
  void*(*start_routine)(void *),
  void *arg
);
```

POSIX Threads Library

Creating a new thread...

A new thread is created via the function pthread_create():

```
#include <pthread.h>

int pthread_create(
   pthread_t *tid,
   const pthread_attr_t *tattr,
   void*(*start_routine)(void *),
   void *arg
);
```

- tid stores the thread ID;
- tattr is used to change the default thread attributes of the newly created thread (often is NULL);
- start_routine is the function executed by the new thread;
- arg refers to the arguments passed to start_routine ;
- pthread_create returns zero when it completes successfully.

POSIX Threads Library

Wait for Thread Termination...

To wait termination of a thread, functions `pthread_join` and `pthread_join` can be used:

```
#include <pthread.h>

int pthread_join(thread_t tid, void **status);
```

POSIX Threads Library

Wait for Thread Termination...

To wait termination of a thread, functions `pthread_join` and `pthread_join` can be used:

```
#include <pthread.h>

int pthread_join(thread_t tid, void **status);
```

The specified thread must be in the current process and must not be detached (see below).

POSIX Threads Library

Wait for Thread Termination...

To wait termination of a thread, functions pthread_join and pthread_join can be used:

```
#include <pthread.h>

int pthread_join(thread_t tid, void **status);
```

The specified thread must be in the current process and must not be detached (see below).

When status is not NULL, it points to a location that is set to the exit status of the terminated thread.

POSIX Threads Library

Thread specific storage. . .

Threads can store private values in thread-specific data (TSD).

POSIX Threads Library

Thread specific storage...

Threads can store private values in thread-specific data (TSD).

In this area, each thread-specific data item is associated with a key that is global to all threads in the process. Using the key, a thread can access a pointer (void $*$) that is maintained per-thread.

POSIX Threads Library

Thread specific storage...

Threads can store private values in thread-specific data (TSD).

In this area, each thread-specific data item is associated with a key that is global to all threads in the process. Using the key, a thread can access a pointer (void *) that is maintained per-thread.

Function pthread_keycreate is called once for each key before the key is used:

```
int pthread_key_create(
  pthread_key_t *key,
  void (*destructor) (void *)
);
```

POSIX Threads Library

Thread specific storage...

Threads can store private values in thread-specific data (TSD).

In this area, each thread-specific data item is associated with a key that is global to all threads in the process. Using the key, a thread can access a pointer (void *) that is maintained per-thread.

Function pthread_keycreate is called once for each key before the key is used:

```
int pthread_key_create(
  pthread_key_t *key,
  void (*destructor) (void *)
);
```

Zero is returned after the operations has been completed successfully. Any other returned value indicates that an error occurred.

POSIX Threads Library

Thread specific storage...

The following functions can be used to manage values in the TSD:

```
int pthread_key_delete(pthread_key_t key);

void *pthread_getspecific(pthread_key_t key);
```

Example: Usage of TSD

POSIX Threads Library

Thread identifier...

The function pthread_self() can be called to return the ID of the calling thread:

```
pthread_t pthread_self(void);
```

POSIX Threads Library

Thread identifier...

The function pthread_self() can be called to return the ID of the calling thread:

```
pthread_t pthread_self(void);
```

Function pthread_equal() can be used to compare two thread ids:

```
int pthread_equal(pthread_t tid1, pthread_t tid2);
```

POSIX Threads Library

Terminating threads...

A thread can terminate its execution in the following ways:

- By returning from its first (outermost) procedure, the threads start routine;
- By calling `pthread_exit()`, supplying an exit status;
- By termination with POSIX cancel functions `pthread_cancel()`.

POSIX Threads Library

Terminating threads...

A thread can terminate its execution in the following ways:

- By returning from its first (outermost) procedure, the threads start routine;
- By calling `pthread_exit()`, supplying an exit status;
- By termination with POSIX cancel functions `pthread_cancel()`.

```
void pthread_exit(void *status)

int pthread_cancel(pthread_t thread)
```

Thread Synchronisation...

We have a race when multiple threads operate on the same set of data.

Thread Synchronisation...

We have a race when multiple threads operate on the same set of data.

To avoid confusion mechanisms to synchronise threads interaction are needed.

Thread Synchronisation...

We have a race when multiple threads operate on the same set of data.

To avoid confusion mechanisms to synchronise threads interaction are needed.

There are a few possible methods of synchronising threads:

- Mutual Exclusion (Mutex) Locks
- Condition Variables
- Semaphores

Thread Synchronisation...

We have a race when multiple threads operate on the same set of data.

To avoid confusion mechanisms to synchronise threads interaction are needed.

There are a few possible methods of synchronising threads:

- Mutual Exclusion (Mutex) Locks
- Condition Variables
- Semaphores

A synchronization objects is a variable used by threads to interact with each other.

Thread synchronisation...

Thread synchronisation is needed when:

- it is the only way to ensure consistency of shared data.
- two or more threads can use a single synchronisation object jointly.
- we have to ensure the safety of mutable data.
- when there is a race.

Mutual Exclusion Locks

Mutual exclusion locks (mutexes) are a common method of serialising thread execution.

Mutual Exclusion Locks

Mutual exclusion locks (mutexes) are a common method of serialising thread execution.

Mutual exclusion locks synchronise threads, usually by ensuring that only one thread at a time executes a critical section of code.

Mutual Exclusion Locks

Mutual exclusion locks (mutexes) are a common method of serialising thread execution.

Mutual exclusion locks synchronise threads, usually by ensuring that only one thread at a time executes a critical section of code.

Mutex locks can also preserve single-threaded code.

Initialising mutexes...

Mutexes are represented by the pthread_mutex_t object.

Initialising mutexes...

Mutexes are represented by the pthread_mutex_t object.

Like most of the objects in the Pthread API, it is meant to be an opaque structure provided to the various mutex interfaces.

Initialising mutexes...

Mutexes are represented by the pthread_mutex_t object.

Like most of the objects in the Pthread API, it is meant to be an opaque structure provided to the various mutex interfaces.

Although you can dynamically create mutexes, most uses are static:

```
/* define and initialize a mutex named 'mutex' */
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Locking mutexes...

Locking (also called acquiring) a Pthreads mutex is accomplished via the pthread_mutex_lock() function:

```
#include <pthread.h>

int pthread_mutex_lock (pthread_mutex_t *mutex);
```

Locking mutexes...

Locking (also called acquiring) a Pthreads mutex is accomplished via the pthread_mutex_lock() function:

```
#include <pthread.h>

int pthread_mutex_lock (pthread_mutex_t *mutex);
```

A successful call to pthread_mutex_lock() will block the calling thread until the mutex pointed at by mutex becomes available.

Releasing mutexes...

The counterpart to locking is unlocking, or releasing, the mutex:

```
#include <pthread.h>

int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

Releasing mutexes...

The counterpart to locking is unlocking, or releasing, the mutex:

```
#include <pthread.h>

int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

A successful call to pthread_mutex_unlock() releases the mutex pointed at by mutex and returns zero.

Releasing mutexes...

The counterpart to locking is unlocking, or releasing, the mutex:

```
#include <pthread.h>

int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

A successful call to pthread_mutex_unlock() releases the mutex pointed at by mutex and returns zero.

The call does not block; the mutex is released immediately.

Example: Using mutexes

```
static pthread_mutex_t the_mutex = PTHREAD_MUTEX_INITIALIZER;


int withdraw (struct account *account, int amount)
{
  pthread_mutex_lock (&the_mutex);
  const int balance = account->balance;
  if (balance < amount) {
    pthread_mutex_unlock (&the_mutex);
    return -1;
  }
  account->balance = balance - amount;
  pthread_mutex_unlock (&the_mutex);
  disburse_money (amount);
  return 0;
}
```

To be continued. . .

GNU Make

Prof. Michele Loreti

Laboratorio di Sistemi Operativi

Corso di Laurea in Informatica (L31)

Scuola di Scienze e Tecnologie

GNU Make

The make utility automates the mundane aspects of building executable from source code.

GNU Make

The make utility automates the mundane aspects of building executable from source code.

make uses a so-called makefile, which contains rules on how to build the executables.

GNU Make

The make utility automates the mundane aspects of building executable from source code.

make uses a so-called makefile, which contains rules on how to build the executables.

You can issue make --help to list the command-line options; or man make to display the man pages.

First Makefile By Example

Let's begin with a simple example to build the Hello-world program (`hello.c`) into executable (`hello`) via `make` utility.

First Makefile By Example

Let's begin with a simple example to build the Hello-world program (hello.c) into executable (hello) via make utility.

```c
// hello.c
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

First Makefile By Example

The following file named makefile contains all the rules needed to build the executable:

```
all: hello

hello: hello.o
	gcc -o hello hello.o

hello.o: hello.c
	gcc -c hello.c

clean:
	rm hello.o hello
```

First Makefile By Example

The following file named makefile contains all the rules needed to build the executable:

```
all: hello

hello: hello.o
	gcc -o hello hello.o

hello.o: hello.c
	gcc -c hello.c

clean:
	rm hello.o hello
```

This file must be in the same directory of your sources.

Makefile syntax. . .

A makefile consists of a set of rules.

Makefile syntax...

A makefile consists of a set of rules.

A rule consists of 3 parts:

- a target,
- a list of pre-requisites
- and a command.

Makefile syntax...

A makefile consists of a set of rules.

A rule consists of 3 parts:

- a target,
- a list of pre-requisites
- and a command.

A rule has the following form:

```
target: pre-req-1 pre-req-2 ...
  command
```

Makefile syntax...

A makefile consists of a set of rules.

A rule consists of 3 parts:

- a target,
- a list of pre-requisites
- and a command.

A rule has the following form:

```
target: pre-req-1 pre-req-2 ...
  command
```

The target and pre-requisites are separated by a colon (:). The command must be preceded by a tab (**NOT spaces!!**).

Makefile syntax. . .

A comment begins with a # and lasts till the end of the line. Long line can be broken and continued in several lines via a back-slash (\).

Makefile syntax...

A comment begins with a # and lasts till the end of the line. Long line can be broken and continued in several lines via a back-slash (\).

A general syntax for the rules is:

```
target1 [target2 ...]: [pre-req-1 pre-req-2 ...]
   [command1
    command2
    .......]
```

Makefile syntax...

A comment begins with a # and lasts till the end of the line. Long line can be broken and continued in several lines via a back-slash (\).

A general syntax for the rules is:

```
target1 [target2 ...]: [pre-req-1 pre-req-2 ...]
   [command1
    command2
    .......]
```

A target that does not represent a file is called a phony target.

Makefile syntax...

A comment begins with a # and lasts till the end of the line. Long line can be broken and continued in several lines via a back-slash (\).

A general syntax for the rules is:

```
target1 [target2 ...]: [pre-req-1 pre-req-2 ...]
   [command1
    command2
    .......]
```

A target that does not represent a file is called a phony target.

If the target is a file, it will be checked against its pre-requisite for out-of-date-ness.

Makefile syntax...

A comment begins with a # and lasts till the end of the line. Long line can be broken and continued in several lines via a back-slash (\).

A general syntax for the rules is:

```
target1 [target2 ...]: [pre-req-1 pre-req-2 ...]
   [command1
    command2
    .......]
```

A target that does not represent a file is called a phony target.

If the target is a file, it will be checked against its pre-requisite for out-of-date-ness.

Phony target is always out-of-date and its command will be run.

Makefile syntax...

A comment begins with a # and lasts till the end of the line. Long line can be broken and continued in several lines via a back-slash (\).

A general syntax for the rules is:

```
target1 [target2 ...]: [pre-req-1 pre-req-2 ...]
   [command1
    command2
    .......]
```

A target that does not represent a file is called a phony target.

If the target is a file, it will be checked against its pre-requisite for out-of-date-ness.

Phony target is always out-of-date and its command will be run.

The standard phony targets are: all, clean, install.

Makefile: Variables

A variable begins with a $ and is enclosed within parentheses ($(CC), $(CC_FLAGS)).

Makefile: Variables

A variable begins with a $ and is enclosed within parentheses ($(CC), $(CC_FLAGS)).

Automatic variables are set by make after a rule is matched. There include:

- $@: the target filename.
- $*: the target filename without the file extension.
- $<: the first prerequisite filename.
- $^: the filenames of all the prerequisites, separated by spaces, discard duplicates.
- $+: similar to $^, but includes duplicates.
- $?: the names of all prerequisites that are newer than the target, separated by spaces.

Example...

```
all: hello

hello: hello.o
    gcc -o $@ $<

hello.o: hello.c
    gcc -c $<

clean:
    rm hello.o hello
```

Virtual paths...

We can use VPATH (uppercase) to specify the directory to search for dependencies and target files.

```
# Search for dependencies and targets from "src" and "include
    " directories
# The directories are separated by space
VPATH = src include
```

Virtual paths...

We can use VPATH (uppercase) to specify the directory to search for dependencies and target files.

```
# Search for dependencies and targets from "src" and "include
    " directories
# The directories are separated by space
VPATH = src include
```

We can also use vpath (lowercase) to be more precise about the file type and its search directory

```
# Search for .c files in "src" directory;
# .h files in "include" directory
# The pattern matching character '%' matches filename without
     the extension
vpath %.c src
vpath %.h include
```

Pattern Rules...

A pattern rule, which uses pattern matching character '%' as the filename, can be applied to create a target, if there is no explicit rule.

```
# Applicable for create .o object file.
# '%' matches filename.
# $< is the first pre-requisite
# $(COMPILE.c) consists of compiler name and compiler options
# $(OUTPUT_OPTIONS) could be -o $@
%.o: %.c
	$(COMPILE.c) $(OUTPUT_OPTION) $<

# Applicable for create executable (without extension)
# from object .o object file
# $^ matches all the pre-requisites (no duplicates)
%: %.o
	$(LINK.o) $^ $(LOADLIBES) $(LDLIBS) -o $@
```

To be continued...