Module 7

Programming With Threads

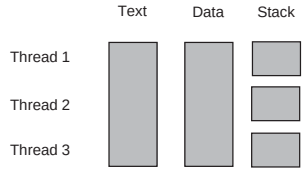
Overview

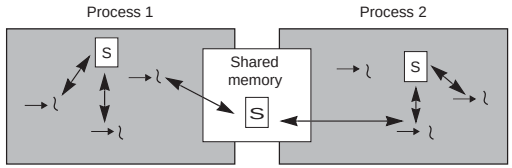
- Objectives
- Relevance

Threads Overview

- Definition
- Benefits
- Synchronization
- Dynamic memory model
- Implementation issues
- Implementation
- Multithreaded architecture

Threads Overview





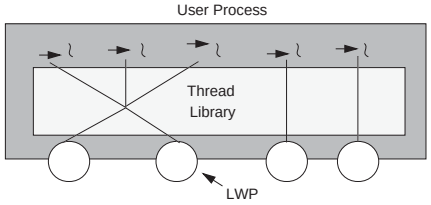
Threads Overview

```
1   /* Some global variables */
2   int count = 0;
3   int stack[10] ;
4
5   /* Two stack operations */
6   void push(int n) {
7     stack[count] = n;
8     count++;
9   }
10
11  int pop() {
12    count--;
13    return stack[count];
14  }
```

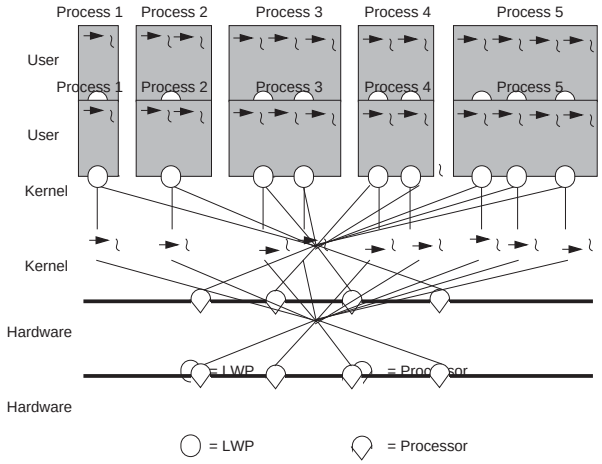

Threads Implementation

- Threads library multiplexes threads on LWPs
- Threads library allows threads to bind to LWPs
- Threads library adjusts number of LWPs required

Threads Implementation



Threads Implementation



Why Have Threads and LWPs?

- Threads
  - Primary, portable programming interface
  - Lighter than LWPs
- LWPs
  - Visible to kernel
  - Real parallelism
  - Complete UNIX functionality
- Best combination of speed, concurrency, functionality, and resource utilization

Creating a Thread

```
1   #include <pthread.h>
2   #define NUM_THREADS 5
3   #define SLEEP_TIME 10
4
5   pthread_t tid[NUM_THREADS]; /* array of thread IDs */
6
7   void *sleeping(void *); /* thread routine */
8
9   void start() {
10    int i;
11
12    for ( i = 0; i < NUM_THREADS; i++)
13      pthread_create(&tid[i], NULL, sleeping,
14          (void *)SLEEP_TIME);
15  }
```


Terminating a Thread

- `pthread_join()`— Examines exit status
- `pthread_exit()`— Terminates itself
- `pthread_cancel()`— Terminates another thread
- Using semaphore locks for notification

Synchronizing Threads

- Working independently
- Synchronization
  - Mutex locks
  - Semaphore locks

Mutual Exclusion

- `pthread_mutex_lock()`— Acquires a mutex and waits for it, if necessary
- `pthread_mutex_trylock()`— Acquires a mutex, if available and returns an error immediately if unavailable
- `pthread_mutex_unlock()`— Releases a mutex

Semaphore Locks

- `sem_wait()`— Acquires or waits for a semaphore by performing a semaphore lock operation
- `sem_trywait()`— Acquires a semaphore by locking the semaphore if not currently locked
- `sem_post()`— Unlocks and increments the count of a semaphore

Summary

Activity	UNIX Mechanism	POSIX Threads Mechanism
Creation	`fork()` and `exec()`	`pthread_create()`
Self-termination	`_exit()`	`pthread_exit()`
Termination by another thread	`kill()`	`pthread_cancel()`
Get unique identifier	`pid = fork()`	`pthread_create(&pid...)`
Mutual exclusion	`fcntl()` and `semop()`	`pthread_mutex_lock()` and `pthread_mutex_unlock()`
Synchronization	`signal()` and `semop()`	`sem_wait()` and `sem_post()`
Change Attributes	`sigaction()` and `nice()`	`pthread_setschedparam()`