Threads

Operating Systems Course
Hebrew University
Spring 2011

What is a Thread?
• A thread lives within a process;
• A process can have several threads.
• A thread possesses an independent flow of control, and can be scheduled to run separately from other threads, because it maintains its own:
  – Stack.
  – Registers. (CPU state)
• The other resources of the process are shared by all its threads.
  – Code
  – Memory
  – Open files
  – And more...

Thread Implementations
• Kernel level threads (lightweight processes):
  – thread management done by the kernel.
• User level threads:
  – kernel unaware of threads.

Kernel Level Threads
• Kernel level threads (lightweight processes)
  – thread management done by the kernel

User Level Threads
• User level threads
  – implemented as a thread library, which contains the code for thread creation, termination, scheduling and switching
  – kernel sees one process and it is unaware of its thread activity.

Implementing a thread library
• Maintain a thread descriptor for each thread
• Switch between threads:
  1. Stop running current thread
  2. Save current state of the thread
  3. Jump to another thread
  • continue from where it stopped before, by using its saved state
• This requires special functions: sigsetjmp and siglongjmp
  – sigset jmp saves the current location, CPU state and signal mask
  – siglong jmp goes to the saved location, restoring the state and the signal mask.
sigsetjmp – save a “bookmark”

sigsetjmp(sigjmp_buf env, int savesigs)
• Saves the stack context and CPU state in env for later use.
• If savesigs is non-zero, saves the current signal mask as well.
• We can later jump to this code location and state using siglongjmp.
• Return value:
  – 0 if returning directly.
  – A user-defined value if we have just arrived here using siglongjmp.

siglongjmp – use a “bookmark”

siglongjmp(sigjmp_buf env, int val)
• Jumps to the code location and restore CPU state specified by env
• The jump will take us into the location in the code where the sigsetjmp has been called.
• If the signal mask was saved in sigsetjmp, it will be restored as well.
• The return value of sigsetjmp after arriving from siglongjmp, will be the user-defined val.

A Demo

• Functions f() and g()
  – Each representing a different thread
• switchThreads()
  – A function that switches between the threads using sigsetjmp and siglongjmp
• main()
  – Initialization and starting the threads.

Demo Code: the threads

```c
void f()
{
  int i=0;
  while(1) {
    ++i;
    printf("in f (%d)\n",i);
    if (i % 3 == 0) {
      printf("f: switching\n");
      switchThreads();
      usleep(SECOND);
    }
  }
}

void g()
{
  ... //similar code
}
```

Demo Code: the switch

```c
sigjmp_buf jbuf[2];

void switchThreads()
{
  static int curThread = 0;
  int ret_val =
    sigsetjmp(jbuf[curThread],1);
  printf("SWITCH: ret_val=%d\n", ret_val);
  if (ret_val == 1) {
    return;
  }
  curThread = 1 - curThread;
  siglongjmp(jbuf[curThread],1);
}
```

The switch

```c
Thread 0:
void switchThreads()
{
  static int curThread = 0;
  int ret_val =
    sigsetjmp(jbuf[curThread],1);
  if (ret_val == 1) {
    return;
  }
  curThread = 1 - curThread;
  siglongjmp(jbuf[curThread],1);
}

Thread 1:
void switchThreads()
{
  static int curThread = 0;
  int ret_val =
    sigsetjmp(jbuf[curThread],1);
  if (ret_val == 1) {
    return;
  }
  curThread = 1 - curThread;
  siglongjmp(jbuf[curThread],1);
}
```
What is saved in `jbuf`?

- Program Counter
  - Location in the code
- Stack pointer
  - Locations of local variables
  - Return address of called functions
- Signal Mask – if specified
- Rest of environment (CPU state)
  - Calculations can continue from where they stopped.

Not Saved:
- Global variables
- Variables allocated dynamically
- Values of local variables
- Any other global resources

Demo Code: initialization

```c
#include <stdio.h>
#include <limits.h>

typedef unsigned long address_t; //64bit address
#define JB_SP 6
#define JB_PC 7

void setup()
{
  unsigned int sp, pc;
  sp = (address_t)stack1 + STACK_SIZE - sizeof(address_t);
  pc = (address_t)f;
  sigsetjmp(jbuf[0],1);
  (jbuf[0]->__jmpbuf)[JB_SP] = translate_address(sp);
  (jbuf[0]->__jmpbuf)[JB_PC] = translate_address(pc);
  sigemptyset(&jbuf[0]->__saved_mask); //empty saved signal mask
  ... //the same for [jbuf[1]] with g
}

int main() {
  setup();
  siglongjmp(jbuf[0],1);
  return 0;
}
```

Implement a user-threads library

- The library should provide thread manipulation functions.
  - Init
  - Spawn
  - Sleep
  - Sync
  - Terminate
  - Get pid
- Library users can create their own threads and use the library functions to manipulate them.
- The library is in charge of thread management and scheduling.

Ex2

Thread State Diagram

The scheduler

- The running thread is always the one with the highest id (i.e. the one created last) compared to all the ready threads.
- If a running thread becomes suspended, the scheduler needs to decide which thread will run instead.
- Use code demos for examples of
  - Thread switching,
  - Using timers and timer signals.
This exercise is difficult, so start early!

Good Luck!