User Thread Implementation

COMP755
Advanced OS
“America is not a blanket woven from one thread, one color, one cloth.”

Jesse Jackson
Thread Implementations

• You can implement the concept of threads at the system level or at the user level.

• The system level has several advantages:
  – Access to all the hardware and CPUs
  – Ability to respond to interrupts
  – Ability to preempt a thread
User Threads

• This is an example of how you might implement threads at the user level.
• It is difficult to preempt a thread. Threads can voluntarily allow another thread to run.
• The yield method allows the thread system to consider scheduling another thread. It does not directly do anything for the calling thread.
Example using User Threads

```c
void main () {
    spawn( printit );
    do {
        ...
        yield( );
        ...
    } until whatever;
}

void printit(void) {
    do {
        print a line;
        yield( );
    } until end of data;
    terminate( );
}
```
Implementation Details

• My example thread system would probably have to be implemented in assembler.
• You have to be able to save the registers and set the stack address
• This user thread implementation does not attempt to use multiple processors
### Thread Control Block (TCB)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return address</td>
<td></td>
</tr>
<tr>
<td>Register save area</td>
<td></td>
</tr>
<tr>
<td>Stack address</td>
<td></td>
</tr>
<tr>
<td>next TCB in a list</td>
<td></td>
</tr>
</tbody>
</table>
Thread System Data

- `ready_list` – a linked list of thread control blocks
- `running` – the currently executing thread’s TCB
private void schedule() { /* internal function */
    Select one of the control blocks from the ready list;
    Set running to the selected thread control block;
    Load the registers from the save area;
    Jump to the return address;
}

• Note that the schedule function does not return to the caller in the usual sense
Starting a New Thread

```java
public void spawn(function) {
    In running thread's control block, save registers and return address;
    Create a new control block;
    Copy the running thread's control block to the new control block;
    Set return address to the address of the function;
    Acquire memory for a stack;
    Put the stack address in the new control block;
    Put return address of terminate() on the stack;
    Put the new control block on the ready list;
    schedule();
}
```
Terminating a Thread

public void terminate() {
    Remove the current TCB from the running list;
    Release the stack memory for the running thread;
    Release the control block for the running thread;
    If no TCBs left on the ready list
        exit(); to terminate the process
    else
        schedule();
}

• Note that the terminate function never returns to the caller
Giving Up Control

• Can be called by a user to allow some other thread the change to run.
• This thread might resume immediately.

```java
public void yield() {
    In running thread's control block, save registers and return address;
    schedule();
}
```
Implement Semaphores

• In a group of no more than three students, write P() and V() methods to implement semaphores

• Turn your implementation in before the end of class

• Include the names of the students in your group
public class Semaphore {
    int counter = 0;
    public Semaphore(int initial) {
        counter = initial;
    }
    public synchronized void V() {
        counter++;
        notify();
    }
    public synchronized void P() {
        while (counter == 0) wait();
        counter--;
    }
}
Semaphore Objects

public class Semaphore {
    int counter;
    TCB waiting; /* list of suspended threads */
    public Semaphore(int initialValue) {
        counter = initialValue;
    }
}

public void P() {
    if (counter <= 0) {
        save registers and return address in TCB;
        remove TCB from ready list;
        put TCB on waiting list;
        schedule();
    }
    counter--;
}
Signal method

public void V() {
    counter++;
    if (waiting != null) {
        Remove one TCB from waiting list;
        put TCB on ready_list;
        yield();
    }
}
}
User and System Threads