P1 - Non-Preemptive Thread Scheduler

CS 4411
Feb 2. 2018
Drew Zagieboylo
The Stack - Review

- a0, a1 -> arguments
- sp -> stack pointer
The Stack - Review

- `a0`, `a1` -> arguments
- `sp` -> stack pointer

```
//funcA
x = 3;
y = 2;
z = add(x, y);
return z;
```
The Stack - Review

- a0, a1 -> arguments
- sp -> stack pointer

```c
//funcA
int z;
x = 3;
y = 2;
foo(x, y, &z);
return z;
```
The Stack - Review

• a0, a1 -> arguments
• sp -> stack pointer
• ra -> return address

```
int z;
x = 3;
y = 2;
foo(x, y, &z);
return z;
```

```
funcA

sp

ra (0xbeec)
```

```
z(?)
x (3)
y (2)
```

pc

sb
The Stack - Review

• a0, a1 -> arguments
• sp -> stack pointer
• ra -> return address

```
int z;
0xbe0   x = 3;
0xbe4   y = 2;
0xbe8   foo(x, y, &z);
0xbeec  return z;
```

```
ra (0xbeec)
  ...
  ...
  ...
```

```
funcA
  z(?)
  x (3)
  y (2)
```

```
sp
  ...
  ...
  ...
```

```
pc -> &foo
sb
```
The Stack - Review

- a0, a1 -> arguments
- sp -> stack pointer
- ra -> return address

```
int z;
0xbe0  x = 3;
0xbe4  y = 2;
0xbe8  foo(x, y, &z);
0xbeec return z;
```

```
sp

funcA

z(?)

x (3)

y (2)

sb

pc

...
The Stack - Review

- Every Process/Thread has its own Stack
- Every Function has its own Stack Frame
- The stack grows down as functions are called; returns go back up the stack
- *Returning* from a function is a *jump* to the *return address* on the stack
Minithreads

- Minithreads are your version of *threads*
- Allows timesharing execution of a single CPU
- Independent stacks, pcs, register values, etc.
- Will need a *Thread Control Block* to manage thread info.
Minithread API

- minithread_fork(process, argument)
  - create a new thread and run process
- minithread_yield()
  - allow a different minithread to execute
- minithread_stop()
  - de-schedule this minithread (stop running until someone explicitly starts you again)
- (a few others that are similar to the above)
Context Switching

• Need to save all current minithread state (registers, pc, sp, etc.)

• Load new minithread’s state from where it was saved, and start executing.

• Where to save state?
  • The stack! (mostly)

• We have provided primitives for context switching:
  • machineprimitives.h/c
  • machineprimitives_x86_86.c/S
mt_switch

- \text{minithread\_switch}(\text{old\_sp}^*, \text{new\_sp}^*)
- saves current processor sp to old_sp
- loads value in new_sp to processor sp
- reloads registers and pc from stack
Context Switching

//proc_1

0xbее0 while (1){
0xbее4
0xbее8  x = x + 1;
0xbееc  mt_yield();
}
Context Switching

//proc_1

0xbee0 while (1){
  x = x + 1;
  mt_yield();
}

0xbee4

0xbee8

0xbeec

pc → &mt_yield
Context Switching

//proc_1

0xbee0 while (1) {
    x = x + 1;
    mt_yield();
}

0xbee4

0xbee8

0xbeec

pc → &mt_yield

proc_1

...
Context Switching

Meanwhile, somewhere else in memory...

```
//proc_1
0xbee0 while (1){
0xbee4    x = x + 1;
0xbee8    mt_yield();
0xbeec }
```

```
pc → &mt_yield
```

```
proc_1
```

```
...saved registers...
```

```
ra(0xbeec)
```

```
...saved registers...
```

```
...tcb_proc1...
```

```
...proc_1...
```
Meanwhile, somewhere else in memory..

tcb_proc1->save sp

//proc_1

0xbee0  while (1){
0xbee4    x = x + 1;
0xbee8    mt_yield();
0xbeec  }

Context Switching
Context Switching

Meanwhile, somewhere else else in memory..

```c
//proc_2

0xfff0 while (1){
0xfff4     x = x - 1;
0xfff8     mt_yield();
0xfffc }   
```

```c
0xfff0 while (1){
0xfff4     x = x - 1;
0xfff8     mt_yield();
0xfffc }   
```

```c
0xfff0 while (1){
0xfff4     x = x - 1;
0xfff8     mt_yield();
0xfffc }   
```

```c
0xfff0 while (1){
0xfff4     x = x - 1;
0xfff8     mt_yield();
0xfffc }   
```
Context Switching

Meanwhile, somewhere else **else** in memory..

```c
while (1) {
    x = x - 1;
    mt_yield();
}
```

```
0xFFF0
0xFFF4
0xFFF8
0xFFFFC
```

```
tcb_proc2->load sp
```

```
//proc_2
```

```
...  ...
...
...
...  ...
```
Context Switching

//proc_1

pc = 0xbee0
pc = 0xbee4
pc = 0xbee8
pc = 0xbeeec

while (1){
    x = x + 1;
    mt_yield();
}

//proc_2

pc = 0xffff0
pc = 0xffff4
pc = 0xffff8
pc = 0xffffc

while (1) {
    x = x - 1;
    mt_yield();
}
Context Switching

- Where do non-running minithreads (tcbs) go?
- A Queue!
- For p1 -> round robin scheduling
Thread Death

- What happens after the return?

```c
void hello_w() {
    printf("Hello World!");
    return;
}
```
void hello_w() {
    printf("Hello World!");
    return;
}
void hello_w() {
    printf("Hello World!");
    return;
}
Thread Death

```c
void hello_w() {
    printf("Hello World!");
    return;
}

sp -> ...

pc = ?
```
void hello_w() {
    printf("Hello World!");
    return;
}

void mt_exit() {
    //do cleanup
    while (1) {};
}
void hello_w() {
    printf("Hello World!");
    return;
}

void mt_exit() {
    //do cleanup
    while (1) {};
}
Thread Death

- Cleanup
  - Free stack
  - Free tcb
- Any problems?
- Need another thread to do cleanup