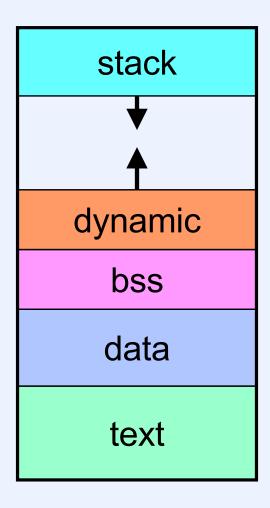
Implementing Threads

The Unix Address Space



Adding More Stuff

stack 1 stack 2 stack 3 mapped file 1 mapped file 2 mapped file 3 mapped file 117 bss & dynamic data text

Subroutines

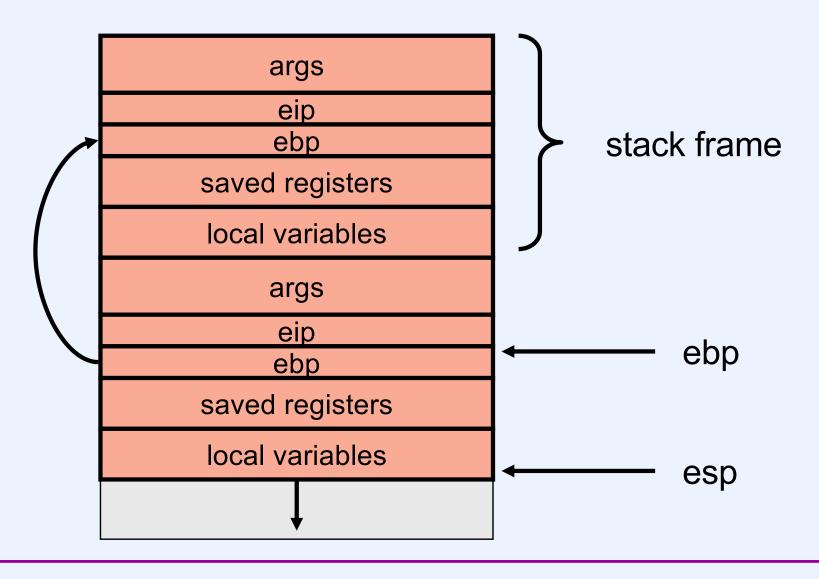
```
int main() {
   int i;
   int a;

   ···

   i = sub(a, 1);
   ···
   return(0);
}
```

```
int sub(int x, int y) {
   int i;
   int result = 1;
   for (i=0; i<y; i++)
      result *= x;
   return(result);
}</pre>
```

Intel x86 (32-Bit): Subroutine Linkage



Intel x86: Subroutine Code (1)

```
movl $0, %eax
main:
                                                           set return
                                     popl %edi
pushl %ebp
                                                           value and
                                     popl %esi
movl %ebp, %esp
movl %esp, %ebp
                                                           restore
                       set up stack
pushl %esi
                                                           frame
                       frame
pushl %edi
                                     popl %ebp
 subl $8, %esp
                                     ret
pushl $1
movl -12 (%ebp), %eax \vdash push args
pushl %eax
call sub
addl $8, %esp
movl %eax, -16(%ebp)
```

Intel x86: Subroutine Code (2)

```
sub:
                                      endloop:
pushl %ebp
                            result = %ecx movl %ecx, -4(%ebp)
movl %esp, %ebp
                            %eax = result movl -4 (%ebp), %eax
subl $8, %esp
                                       movl %ebp, %esp
movl $1, -4(%ebp) initialize result
                                       popl %ebp
movl $0, -8(%ebp) initialize i
                                       ret
movl -4 (%ebp), %ecx %ecx holds result
movl -8 (%ebp), %eax %eax holds i
beginloop:
cmpl 12(%ebp), %eax y:i
jge endloop
imull 8 (%ebp), %ecx result *= x
addl $1, %eax
                      i++
jmp beginloop
```

x86-64

- Twice as many registers
- Arguments may be passed in registers, rather than on stack
- No special-purpose base pointer
 - use stack pointer instead

Intel x86-64: Subroutine Code (1)

```
main:
    subq $24, %rsp  # reserve space on stack for locals
...
    movl 12(%rsp), %edi  # set first argument
    movl $1, %esi  # set second argument
    call sub
    addl $24, %rsp
...
    movl $0, %eax  # set return value
    ret
...
```

II–9

Intel x86-64: Subroutine Code (2)

```
sub:
testl %esi, %esi  # leaf function: no stack setup
jle skiploop
movl $1, %eax
movl $0, %edx
loop:
imull %edi, %eax
addl $1, %edx
cmpl %esi, %edx
jne loop
ret
skiploop:
movl $1, %eax
ret
```

SPARC Architecture

	_		_	
return address	i7 r31		о7	r15
frame pointer	i6 r30	stack pointer	06	r14
·	i5 r29	•	05	r13
	i4 r28		04	r12
	i3 r27		03	r11
	i2 r26		03	r10
			1	
	i1 r25		01	r9
	i0 r24		00	r8
Input Registers		Output Registers		
] 17 r23		g 7	r7
	l6 r22		g6	r6
	15 r21			
			g5	r5
	l4 r20		g4	r4
	l3 r19		g3	r3
	l2 r18		g2	r2
	l1 r17		g1	r1
	l0 r16	0	g0	r0
Local Registers		Global Registers	-	

SPARC Architecture: Register Windows

input window 1 local output input local window 2 input output window 3 local output

SPARC Architecture: Stack

storage for local variables

dynamically allocated stack space

space for compiler temporaries and saved floating point registers

outgoing parameters beyond 6th

save area for callee to store register arguments

one-word "hidden" parameter

16 words to save in and local regs

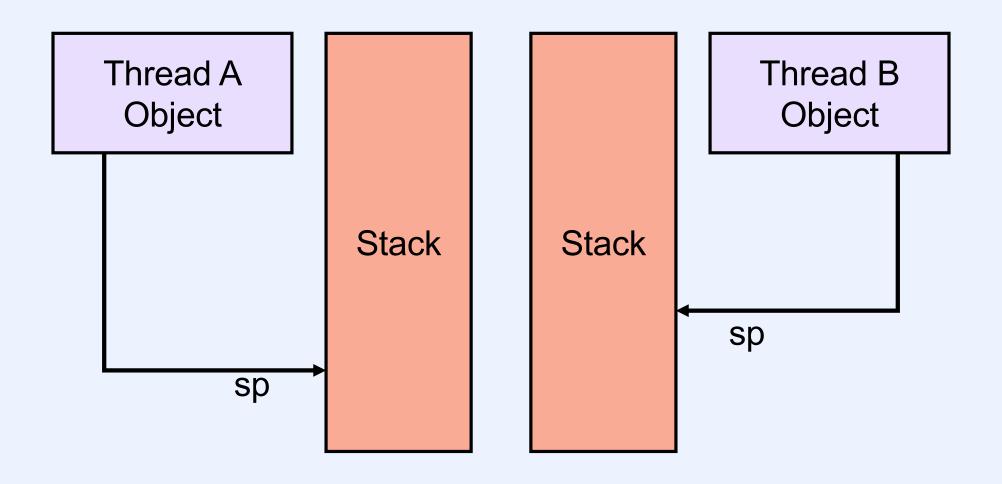
FP, old SP

---- SF

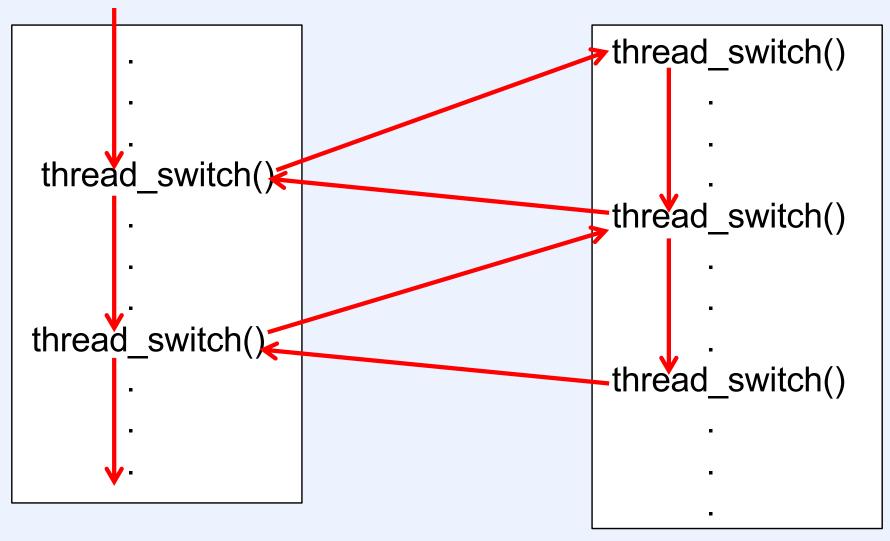
SPARC Architecture:Subroutine Code

```
sub:
ld [%fp-8], %o0
                              save %sp, -64, %sp
   ! put local var (a)
                                 ! push a new
   ! into out register
                                 ! stack frame
mov 1, %o1
                              add %i0, %i1, %i0
   ! deal with 2nd
                                 ! compute sum
   ! parameter
                              ret
call sub
                                 ! return to caller
nop
                              restore
st %o0, [%fp-4]
                                 ! pop frame off
                                 ! stack (in delay slot)
   ! store result into
   ! local var (i)
```

Representing Threads



Switching Between Threads



Coroutine linkage

Switching Between Threads

```
1 void thread_switch(thread_t *next_thread) {
2    getcontext(&CurrentThread->ctx);
3    CurrentThread = next_thread;
4    setcontext(&CurrentThread->ctx);
5    return;
6 }
```

Switching Between Threads, Take 2

```
void thread_switch(thread_t *next_thread) {
volatile int first = 1;
getcontext(&CurrentThread->ctx);

if (first) {
    first = 0;
    CurrentThread = next_thread;
    setcontext(&CurrentThread->ctx);
}

return;
}
```

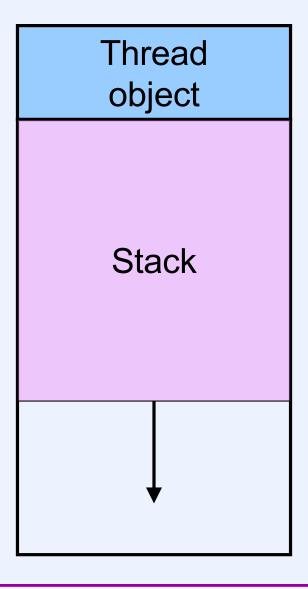
Quiz 1

```
void thread switch(thread t *next thread) {
      volatile int first = 1;
      getcontext(&CurrentThread->ctx);
      if (first) {
5
           first = 0;
           CurrentThread = next thread;
           setcontext(&CurrentThread->ctx);
8
      return;
10
     Does this implementation of thread switch
     work?
       a) yes: in all cases
       b) yes, except for a few edge cases
       c)
          no
```

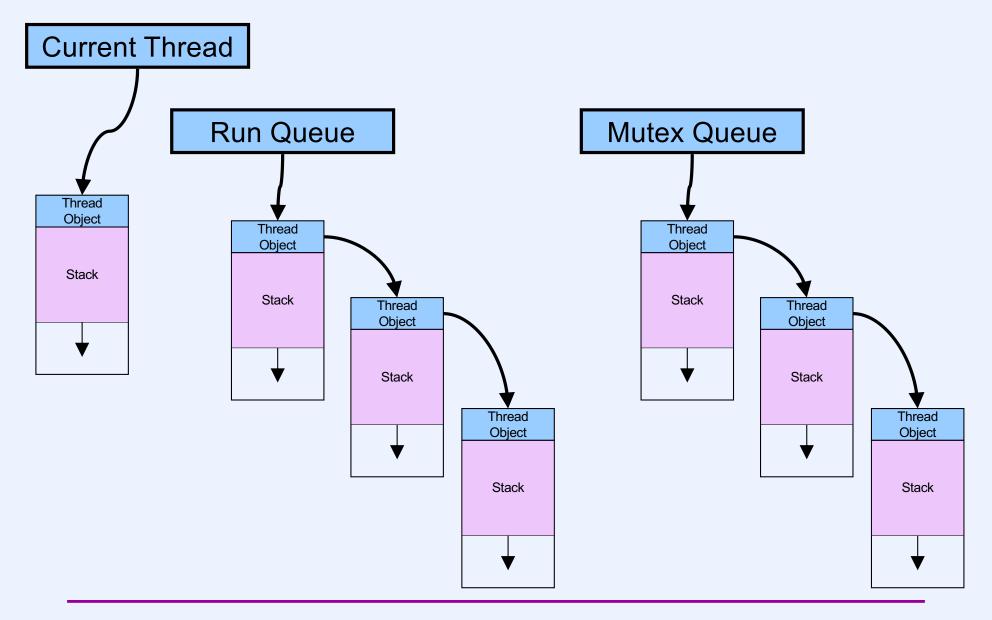
A Simple Threads Implementation

- Basis for user-level threads package
- Straight-threads implementation
 - no interrupts
 - everything in thread contexts
 - one processor

Basic Representation

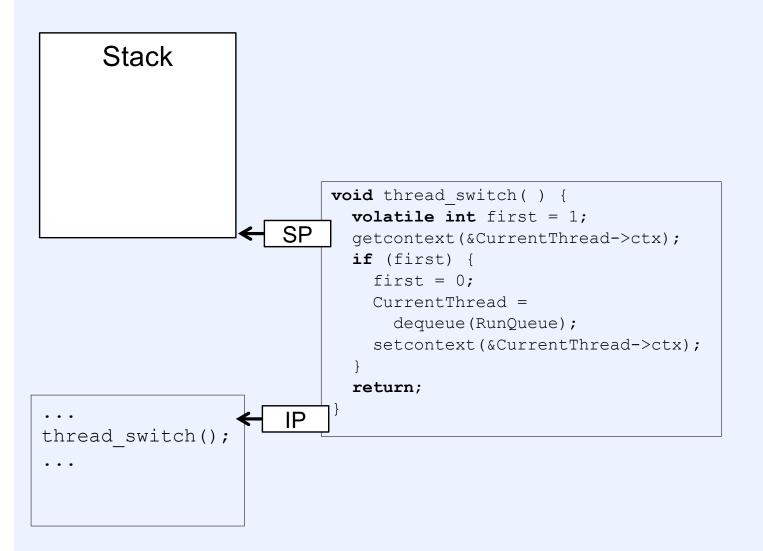


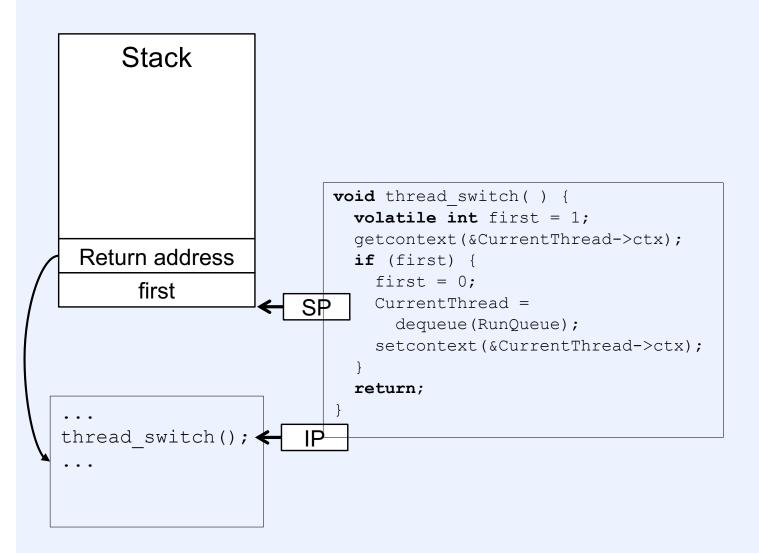
A Collection of Threads

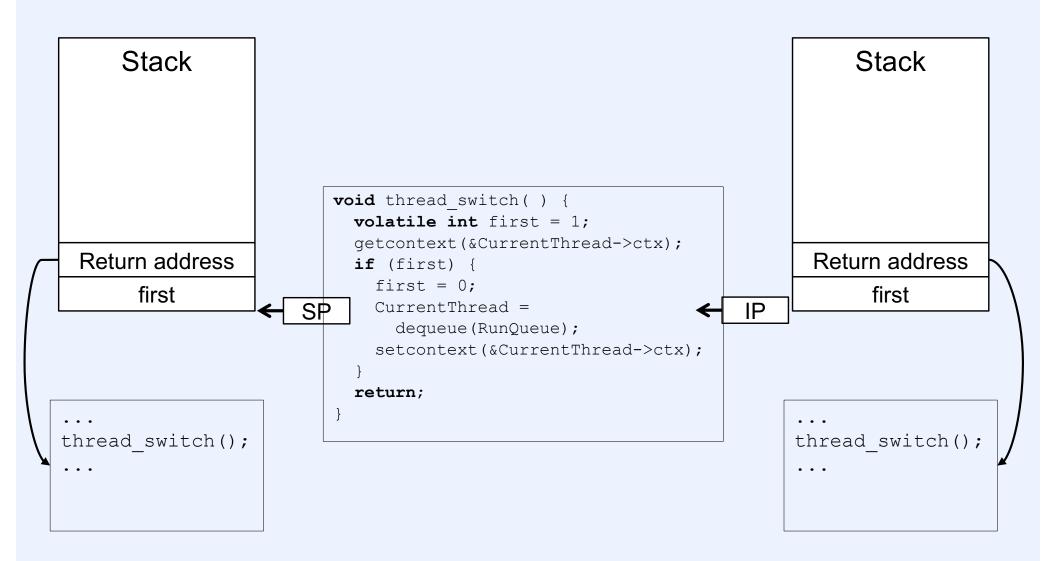


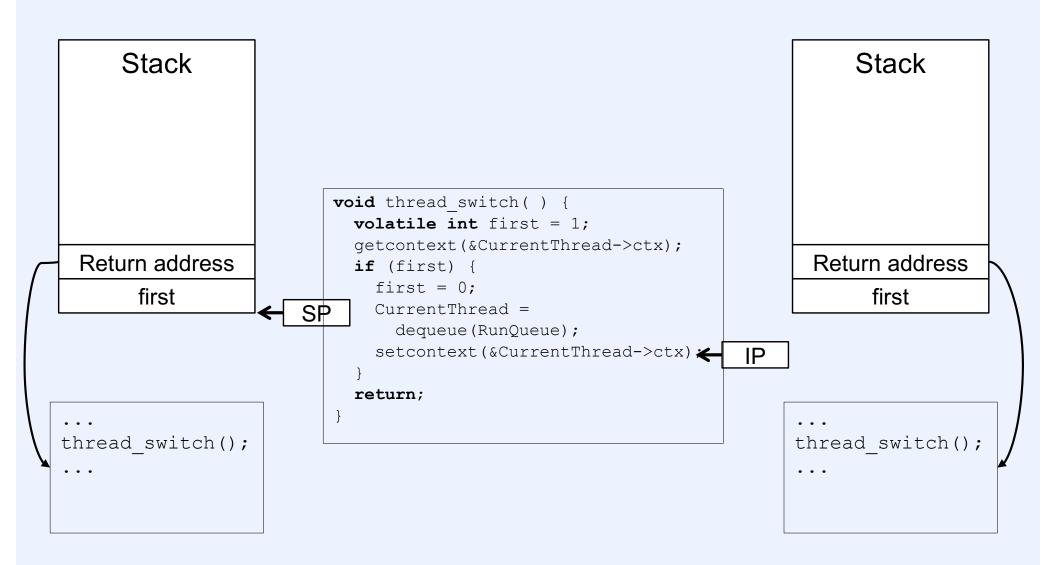
Thread Switch (using Run Queue)

```
void thread_switch() {
   volatile int first = 1;
   getcontext(&CurrentThread->ctx);
   if (first) {
      first = 0;
      CurrentThread = dequeue(RunQueue);
      setcontext(&CurrentThread->ctx);
   }
   return;
}
```









Stack Return address first thread switch();

```
Stack
void thread switch() {
 volatile int first = 1;
  getcontext(&CurrentThread->ctx);
                                               Return address
 if (first) {
    first = 0;
                                                     first
    CurrentThread =
      dequeue (RunQueue);
    setcontext(&CurrentThread->ctx);
  return;
                                        IP
                                             thread switch();
```

```
Stack
void thread switch() {
 volatile int first = 1;
  getcontext(&CurrentThread->ctx);
 if (first) {
    first = 0;
    CurrentThread =
      dequeue (RunQueue);
    setcontext(&CurrentThread->ctx);
  return;
                                             thread switch();
```

Mutexes

```
mutex_t mut;

mutex_lock(&mut);

x = x+1;

mutex_unlock(&mut);
```

Implementing Mutexes

```
void mutex lock(mutex_t *m) {
  if (m->locked) {
    enqueue (m->wait queue, CurrentThread);
    thread switch();
 m->locked = 1;
void mutex unlock(mutex_t *m) {
 m->locked = 0;
  if (!queue empty(m->wait queue))
    enqueue (RunQueue, dequeue (m->wait queue));
```

Quiz 2

```
void mutex lock(mutex t *m) {
  if (m->locked) {
    enqueue (m->wait queue, CurrentThread);
    thread switch();
  m->locked = 1;
void mutex unlock(mutex t *m) {
  m->locked = 0;
  if (!queue empty(m->wait queue))
    enqueue (RunQueue, dequeue (m->wait queue));
```

- a) It works.
- b) It works as long as there are no more than two threads.
- c) There are situations in which it doesn't work for any number of threads greater than 1.

Implementing Mutexes, Take 2

```
void mutex lock(mutex_t *m) {
  if (m->locked) {
    enqueue (m->queue, CurrentThread);
    thread switch();
  } else
    m->locked = 1;
void mutex unlock(mutex_t *m) {
  if (queue empty(m->queue))
    m->locked = 0;
  else
    enqueue (runqueue, dequeue (m->queue));
```

Thread Termination

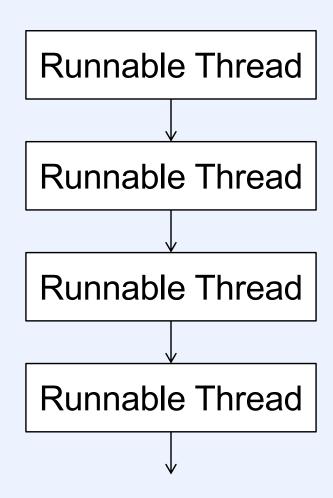
- Termination
 - thread becomes zombie
 - if joinable
 - notify waiter, if present
 - if detached
 - disappear
 - thread can't do this by itself!

The Reaper Thread

```
while(1) {
    wait_for_terminated_zombie()
    delete(zombie);
}
```

Thread Yield

Current Thread



Thread Yield Details

```
void thread_yield() {
   if (!queue_empty(runqueue)) {
     enqueue(runqueue, CurrentThread);
     thread_switch();
   }
}
```

Time Slicing

- Periodically
 - current thread forced to do a thread yield

```
void ClockInterrupt(int sig) {
   thread_yield();
}
```

Implement ClockInterrupt with VTALRM signal