PThreads Lab
Review

What needs synchronized?

What happens without synchronization?

```
void* runner(void* param)
{
  int tid = *(int*) param;
  int i;

  for(i=0 ; i < 100 ; i++)
  {

    //update the global counter
    global += .1;
    fprintf(stderr,"[%d] I have the mutex! %f\n",
       tid, global);

    //track how many times each thread updates the
    //global variable
    gThreadExecutionCouter[tid] ++;
  }

  return NULL;
}
```

gcc -o bin/CS460_PthreadsSimple.s -S src/CS460_PthreadsSimple.c

```
// move double precision floating point to
// register xmm1 (SSE2 register) SSE2 (Streaming SIMD
// Extenstion 2) (Single Instruction Multiple Data)

// global is a label output by gcc to make the code
// more readable
movsd   global(%rip), %xmm1

// move constant to register xmm0
// constant is stored in .rodata
movsd   .LC0(%rip), %xmm0

// add xmm0 = xmm0 + xmm1
addsd   %xmm1, %xmm0

//store xmm0 back to global
movsd   %xmm0, global(%rip)
```

Bottom of the file

```
	.section	.rodata
	.align 8
.LC0:
	.long	2576980378
	.long	1069128089
	.ident	"GCC: (SUSE Linux) 4.5.0 20100604 [gcc-4_5-branch revision
	 160292]"
	.section	.comment.SUSE.OPTs,"MS",@progbits,1
	.string "ospwg"
	.section	.note.GNU-stack,"",@progbits
```

http://www.math.okstate.edu/~yqwang/teaching/math4513_fall11/Notes/0point1.pdf

Assembly

```
movsd  global(%rip), %xmm1

%rip

global(%rip) means
```

```
int globalInt; // not a double

movl globalInt(%rip), %eax

addl $1, %eax

movl %eax, globalInt(%rip)
```

```
                          for(i = 0; i < numbThreads; i++)
                          {
                            pThreadId = (int*) malloc(sizeof(int));
                            *pThreadId = i;
                            gThreadExecutionCouter[*pThreadId] = 0;     // &i
                            pthread_create(&(tids[i]), &attr, runner, pThreadId);
                          }
```

```
void* runner(void* param)
{
  int tid = *(int*) param;
  int i;

  for(i=0 ; i < 100 ; i++)
  {

    //update the global counter
    global += .1;
    fprintf(stderr,"[%d] I have the mutex! %f\n",
       tid, global);

    //track how many times each thread updates the
    //global variable
    gThreadExecutionCouter[tid] ++;
  }

  return NULL;
}
```

Why is the &i a problem?

Where is the memory leak?
How do we fix it?

Threads vs Processes

- What impacts the number of threads you can spawn?

- What impacts timing the pthread_create() or fork() commands?

[1]http://dustycodes.wordpress.com/2012/02/09/increasing-number-of-threads-per-process/

Thread vs Process

```
struct timespec gsStart;
struct timespec gsStop;
unsigned long gTimer = 0;

void* runner(void* param)
{
  clock_gettime(CLOCK_REALTIME, &gsStop);
  gTimer = (gsStop.tv_sec * BILLION + gsStop.tv_nsec )
           - (gsStart.tv_sec * BILLION + gsStart.tv_nsec);
  printf(“%d\n”, gTimer);

  return NULL;
}

int main()
{
  loop
    clock_gettime(CLOCK_REALTIME, &gsStart);
    pthread_create(&tid, &attr, runner, NULL);
    clock_gettime(CLOCK_REALTIME, &stop);
    pthread_join(tid, NULL);
    timer = (stop.tv_sec * BILLION + stop.tv_nsec ) -(gsStart.tv_sec *
        BILLION + gsStart.tv_nsec );
    fprintf(stderr,"%d\n",timer);
```

32000 processes or threads. Max number of fork()s I could successfully run.

What does the fork() test look like?

Problems?

Results

Threads		Processes		
runner	Main	Child	Parent	
0.842895070	0.551234762	3.983038903	3.164800334	
0.833103081	0.549708274	5.207769598	4.217674424	
0.816265136	0.536787647	4.712554553	3.879002262	
0.834968428	0.541243202	4.804507226	3.994651887	
0.804357838	0.535688139	3.967425267	3.310980898	
0.827102189	0.538802759	3.791980938	3.132581683	
0.822940227	0.536047850	4.654110030	3.949547746	
0.820799335	0.546865884	4.658301847	3.991315702	
0.827575525	0.541951887	3.938950736	3.358846209	
0.833165043	0.543715171	3.849118452	3.146622925	
0.826317187	0.542204558	4.356775755	3.614602407	Average
0.827338857	0.541597545	4.318574467	3.618924236	Median

Total time, in seconds, to launch 32000 threads or processes

Game Of Life

Questions?

Synchronization data types

- pthread_mutex_t
- pthread_cond_t
- pthread_rwlock_t