19CS2106S

Operating Systems Design

Session: 32

Solution that uses the exchange primitive to build a lock

```
/* Example: Creating a Thread and waiting for Thread Completion */
#include <stdio.h>
 #include <pthread.h>
 #include <stdlib.h>

 typedef struct __myarg_t {
     int a;
     int b;
 } myarg_t;

 typedef struct __myret_t {
     int x;
     int y;
 } myret_t;

 void *mythread(void *arg) {
     myarg_t *m = (myarg_t *) arg;
     printf("%d %d\n", m->a, m->b);
     myret_t *r = malloc(sizeof(myret_t));
     r->x = 1;
     r->y = 2;
     return (void *) r;
 }
int main(int argc, char *argv[]) {
     int rc;
     pthread_t p;
     myret_t *m;

     myarg_t args;
     args.a = 10;
     args.b = 20;
     pthread_create(&p, NULL, mythread, &args);
     pthread_join(p, (void **) &m);  // this thread has been waiting inside of the
pthread_join() routine.
     printf("returned %d %d\n", m->x, m->y);
     return 0;
 }
/*
[vishnu@team-osd ~]$ cc thread.c -lpthread
[vishnu@team-osd ~]$ ./a.out
10 20
returned 1 2
*/

/* Example: Simpler Argument Passing to a Thread */
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

void *mythread(void *arg) {
     int m = (int) arg;
     printf("%d\n", m);
     return (void *) (arg + 1);
 }

 int main(int argc, char *argv[]) {
     pthread_t p;
     int rc, m;
     pthread_create(&p, NULL, mythread, (void *) 100);
     pthread_join(p, (void **) &m);
     printf("returned %d\n", m);
```

```
     return 0;
 }
/*
[vishnu@team-osd ~]$ cc threadsinglevalue.c -lpthread
threadsinglevalue.c: In function ‘mythread’:
threadsinglevalue.c:7:14: warning: cast from pointer to integer of different size [-
Wpointer-to-int-cast]
      int m = (int) arg;
              ^
[vishnu@team-osd ~]$ ./a.out
100
returned 101
*/

/* Why it gets worse while Shared Data, nondeterministic start-up and uncontrolled
scheduling - A solution that uses the exchange primitive to build a lock. main-thread-
5.c */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int max;
volatile int balance = 0;

//
// xchg(int *addr, int newval)
// return what is pointed to by addr
// at the same time, store newval into addr
//
static inline uint
xchg(volatile unsigned int *addr, unsigned int newval)
{
    uint result;
    asm volatile("lock; xchgl %0, %1" : "+m" (*addr), "=a" (result) : "1" (newval) :
"cc");
    return result;
}

volatile unsigned int mutex = 0;

void
SpinLock(volatile unsigned int *lock) {
    while (xchg(lock, 1) == 1)
      ; // spin
}

void
SpinUnlock(volatile unsigned int *lock) {
    xchg(lock, 0);
}


void *
mythread(void *arg)
{
    char *letter = arg;
    //cpubind();
    printf("%s: begin\n", letter);
    int i;
    for (i = 0; i < max; i++) {
      SpinLock(&mutex);
      balance = balance + 1;
      SpinUnlock(&mutex);
    }
    printf("%s: done\n", letter);
    return NULL;
}
```

```
int
main(int argc, char *argv[])
{
    if (argc != 2) {
      fprintf(stderr, "usage: main-first <loopcount>\n");
      exit(1);
    }
    max = atoi(argv[1]);

    pthread_t p1, p2;
    printf("main: begin [balance = %d]\n", balance);
    pthread_create(&p1, NULL, mythread, "A");
    pthread_create(&p2, NULL, mythread, "B");
    // join waits for the threads to finish
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf("main: done\n [balance: %d]\n [should: %d]\n",
         balance, max*2);
    return 0;
}
/*
[vishnu@localhost threads]$ cc main-thread-5.c -lpthread
[vishnu@localhost threads]$ ./a.out
usage: main-first <loopcount>
[vishnu@localhost threads]$ ./a.out 4
main: begin [balance = 0]
A: begin
B: begin
B: done
A: done
main: done
 [balance: 8]
 [should: 8]
*/
```