ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Διαδιεργασιακή Επικοινωνία
Εργαστηριακές Ασκήσεις

Υλικό από:
Modern Operating Systems Laboratory Exercises, Shrivakan Mishra

Σύνθεση
Κ.Γ. Μαργαρίτης, Τμήμα Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας

1. Εκτελέστε τα animations που σχετίζονται με την ενότητα από την ιστοσελίδα του Εκπαιδευτικού Υλικού στο http://www.it.uom.gr/teching.html/

2. 'Εκτελέστε' με χαρτί και μολύβι διάφορα σενάρια για τα απλά παραδείγματα που παρουσιάζονται στις παραδόσεις (ΟΧΙ τα πλήρη προγράμματα, αλλά τα αποσπάσματα κώδικα ή ψευδοκώδικα που δείχνουν τις αρχές λειτουργίας των διαφόρων τεχνικών). Βεβαιωθείτε οτι λαμβάνετε υπ' όψη σας τις χειρότερες περιπτώσεις κατά την εκτέλεση, όχι τις καλύτερες.

3. Εκτελέστε τα προγράμματα C και Java. Σε μερικά προγράμματα C λείπουν οι κατάλληλες δηλώσεις include. Σε μερικά προγράμματα Java λείπει η κλάση main. Προσπαθήστε να να συμπληρώσετε.

4. Μελετήστε και εκτελέστε τα προγράμματα που ακολουθούν. Προσπαθήστε να τροποποιήσετε τον κώδικα έτσι ώστε να ελέγχετε το ρυθμό παραγωγής-κατανάλωσης στοχείων.

5. Βρείτε στο διαδίκτυο αντίστοιχα προγράμματα σε C και Java για τις τεχνικές concurrent programming που συζητήσαμε. Συλλέξτε κώδικα ή links.

6. Προσπαθήστε να βρείτε, είτε από το MSDN είτε από αλλού στο διαδίκτυο, αντίστοιχα προγράμματα σε C με κλήσεις συστήματος Win32. Συλλέξτε κώδικα ή links.

```
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#define BUFF_SIZE 4
#define SHARED 1

char buffer[BUFF_SIZE];
int nextIn = 0;
int nextOut = 0;

sem_t empty_slots;
sem_t full_slots;

void Put(char item) {

        sem_wait(&empty_slots);
        buffer[nextIn] = item;
        nextIn = (nextIn + 1) % BUFF_SIZE;
        printf("Producing %c ...\n", item);
        sem_post(&full_slots);

}

void * Producer() {

        int i;
        for(i = 0; i < 10; i++)
                Put((char)('A'+ i % 26));

}
```

```
void Get() {

        int item;
        sem_wait(&full_slots);
        item = buffer[nextOut];
        nextOut = (nextOut + 1) % BUFF_SIZE;
        printf("Consuming %c ...\n", item);
        sem_post(&empty_slots);

}

void * Consumer() {

        int i;
        for(i = 0; i < 10; i++)
                Get();

}


main() {

        pthread_t tid1, tid2;

        sem_init(&empty_slots, SHARED, 4);
        sem_init(&full_slots, SHARED, 0);
        pthread_create(&tid1, NULL, Producer, NULL);
        pthread_create(&tid2, NULL, Consumer, NULL);
        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);

}
```

```
/* Producer/consumer program illustrating conditional variables */
#include<pthread.h>
#include <stdio.h>

#define BUF_SIZE 3                                              /* Size of shared buffer */

int buffer[BUF_SIZE];                                           /* shared buffer */
int add=0;                                                      /* place to add next element */
int rem=0;                                                      /* place to remove next element */
int num=0;                                                      /* number elements in buffer */
pthread_mutex_t m=PTHREAD_MUTEX_INITIALIZER     ;       /* mutex lock for buffer */
pthread_cond_t c_cons=PTHREAD_COND_INITIALIZER; /* consumer waits on this cond var */
pthread_cond_t c_prod=PTHREAD_COND_INITIALIZER; /* producer waits on this cond var */

void *producer(void *param);
void *consumer(void *param);

main (int argc, char *argv[])
{
        pthread_t tid1, tid2;           /* thread identifiers */
        int i;

        /* create the threads; may be any number, in general */
        if (pthread_create(&tid1,NULL,producer,NULL) != 0) {
                fprintf (stderr, "Unable to create producer thread\n");
                exit (1);
        }
        if (pthread_create(&tid2,NULL,consumer,NULL) != 0) {
                fprintf (stderr, "Unable to create consumer thread\n");
                exit (1);
        }
        /* wait for created thread to exit */
        pthread_join(tid1,NULL);
        pthread_join(tid2,NULL);
        printf ("Parent quiting\n");
}
```

```
/* Produce value(s) */
void *producer(void *param)
{
        int i;
        for (i=1; i<=20; i++) {                                 /* Insert into buffer */
                pthread_mutex_lock (&m);
                if (num > BUF_SIZE) exit(1);                    /* overflow */
                while (num == BUF_SIZE)                         /* block if buffer is full */
                        pthread_cond_wait (&c_prod, &m);                /* if executing here, buffer not full so add element */
                buffer[add] = i;
                add = (add+1) % BUF_SIZE;
                num++;
                pthread_mutex_unlock (&m);
                pthread_cond_signal (&c_cons);
                printf ("producer: inserted %d\n", i);  fflush (stdout);
        }
        printf ("producer quiting\n");  fflush (stdout);
}

/* Consume value(s); Note the consumer never terminates */
void *consumer(void *param)
{
        int i;
        while (1) {
                pthread_mutex_lock (&m);
                if (num < 0) exit(1);                                   /* underflow */
                while (num == 0)                                /* block if buffer empty */
                        pthread_cond_wait (&c_cons, &m);                /* if executing here, buffer not empty so remove
element */
                i = buffer[rem];
                rem = (rem+1) % BUF_SIZE;
                num--;
                pthread_mutex_unlock (&m);
                pthread_cond_signal (&c_prod);
                printf ("Consume value %d\n", i);  fflush(stdout);
        }
}
```