ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Διεργασίες και Νήματα
Εργαστηριακές Ασκήσεις

Υλικό από:
Modern Operating Systems Laboratory Exercises, Shrivakan Mishra

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

1. Εκτελέστε σε περιβάλλον Linux τα προγράμματα C που ακολουθούν.

2. Στο πρόγραμμα του slide 4: Τροποποιείστε το ώστε οι δύο διεργασίες να εκτελούν ls σε διαφορετικούς καταλόγους. Οι κατάλογοι να δίνονται ως όρισμα απο τη γραμμή εντολών.

3. Ποιές είναι οι διαφορές στα προγράμματα των slides 5 και 6; Τροποποιείστε το πρόγραμμα 4 ώστε να έχουμε ένα for loop που να δημιουργεί 100 διεραγσίες. Τροποποιείστε το πρόγραμμα στο slide 5 ώστε οι δημιουργίες διεργασιών-παιδιών να είναι εμφωλευμένες/αναδρομικές.

4. Με τη βοήθεια του προγράμματος στα slides 7 και 8 δημιουργείστε ένα πρόγραμμα που να καλεί δικά σας προγράμματα και σενάρια φλοιού με τη βοήθεια της γραμμής εντολών.

5. Τροποποιείστε το πρόγραμμα στο slide 9 ώστε ο φλοιός μας να μπορεί να επεξεργαστεί εντολές με στοιχειώδη ορίσματα.

6. Στο πρόγραμμα των slides 10 και 11, τι θα αλλαζε αν θέλαμε να γράψουμε ένα πρόγραμμα που να υλοποεί την εντολή mv ; Αν αλλάξουμε το BLOCK_SIZE σε 32 ή 128 ή 512 μπορούμε να δούμε κάποια διαφορά στην αντιγραφή ενός αρχείου 2MB;

7. Στο πρόγραμμα του slide 12: Συγκρίνετε τη δημιουργία 10 νημάτων με αυτή 10 διεργασιών. Διαπιστώστε τη διαφορά στην εκτέλεση μέσω των εντολων top, ps κλπ.

8. Τι κάνει το πρόγραμμα των slides 13 και 14; Τροποποιείστε το πρόγραμμα ώστε να έχουμε ένα πολυνηματικό copy απο ένα αρχείο σε ένα άλλο.

9. Τι κάνει το πρόγραμμα των slides 15 και 16; Τροποποιείστε το πρόγραμμα ώστε η μεταβλητή result να είναι καθολική και να αρχικοποιείται στην αρχή του προγράμματος. Επίσης μεσα στη BusyWork η σχέση γίνεται result = result + 1; Τι διαπιστώνουμε στο αποτέλεσμα;

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
  int pid, status;

  printf("Forking process\n");
  pid = fork();
  wait(&status);
  printf("The process id is %d \n", getpid());
  printf("value of pid is %d\n", pid);
  if (pid == 0)
    printf("I am the child, status is %d\n", status);
  else
    printf("I am the parent, status is %d\n", status);
  execl("/bin/ls","/bin/ls","-l", NULL);
  printf("This line is not printed\n");
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main(int argc, char *argv[])
{
     int i;
     pid_t h;          /* δήλωση της μεταβλητής h τύπου pid_t */
     h = fork();       /* δημιούργησε αντίγραφο               */
     if (h==0) {       /* αυτό εκτελείται από το παιδί        */
            for (i=0; i<3; i++) {
                 printf("**Child %d\n", i);
            }
            exit(0);   /* το παιδί τερματίζει κανονικά        */
     }
     else if (h>0) { /* αυτό εκτελείται από τον γονέα         */
            for (i=0; i<3; i++) {
                 printf("*Parent %d\n", i);
            }
            exit(0);   /* ο γονέας τερματίζει κανονικά        */
     }
     else {            /* δεν δημιουργήθηκε παιδί             */
            printf("fork error!\n");
            exit(1);   /* ο γονέας τερματίζει με σφάλμα       */
     }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main(int argc, char *argv[])
{
     pid_t h1, h2, h;
     int status, i;
     h1 = fork();              /* Δημιουργία πρώτου παιδιού */
     if (h1 < 0) {
          printf("Error forking process 1\n");
          exit(1);             /* ο γονέας τερματίζει με σφάλμα */
     }
     else if (h1 == 0) {
                               /* Κώδικας παιδιού 1 */
          exit(0);             /* το παιδί 1 τερματίζει κανονικά */
     }
     h2 = fork();              /* Δημιουργία δεύτερου παιδιού */
     if (h2 < 0) {
          printf("Error forking process 2\n");
          exit(1);             /* ο γονέας τερματίζει με σφάλμα */
     }
     else if (h2 == 0) {
                               /* Κώδικας παιδιού 2 */
          exit(0);             /* το παιδί 2 τερματίζει κανονικά */
     }
                               /* Από εδώ και κάτω κώδικας γονέα: */
     h = wait(&status);        /* περίμενε ένα οποιοδήποτε παιδί */
     printf("Parent message: Child %d has finished\n", h);
     h = wait(&status);        /* περίμενε το δεύτερο παιδί */
     printf("Parent message: Child %d has finished\n", h);
     exit(0);                  /* ο γονέας τερματίζει κανονικά */
}
```

```
/* myprogram.c: παίρνει ένα όρισμα n θετικό ακέραιο            */
/* από τη γραμμή εντολών και τυπώνει ένα τρίγωνο με n γραμμές */
/* gcc -o myprogram myprorgram.c                               */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
       int i,j;
       printf("%d", argc);
       if (argc != 2) {
           printf("Usage: %s a positive integer\n", argv[0]);
           exit(0);
       }
       for (i=0; i<atoi(argv[1]); i++){
           for (j=1; j<=i; j++)
               printf("*");
           printf("\n");
       }
}
```

```
/* test.c εκτελεί το "myprogram 25" και "myprogram 38"    */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main(int argc, char *argv[]) {
        pid_t h1, h2;
        h1 = fork();                /* δημιούργησε ένα παιδί */
        if (h1<0) {exit(1);}        /* error forking child 1 */
        if (h1==0) {
                                    /* αντικατέστησε το παιδί*/
                        /* με το πρόγραμμα "myprogram 25" */
              execl("./myprogram","./myprogram", "25", NULL);
        }
        h2 = fork();            /* δημιούργησε δεύτερο παιδί  */
        if (h2<0) {exit(1);}        /* error forking child 2  */
        if (h2==0) {
                        /* αντικατέστησε το δεύτερο παιδί  */
                        /* με το πρόγραμμα "myprogram 38"  */
             execl("./myprogram","./myprogram", "38", NULL);
        }
}
```

```
#include <stdio.h>
#include <unistd.h>

int main(void) {
 char command[80];
 int pid, status;
 for(;;) {
  printf("my_shell: ");
  if(fgets(command, sizeof(command), stdin) == NULL) {
   printf("\n");
   return 0;
  }
  command[strlen(command) - 1] = '\0';

  if((pid = fork()) == 0)
   execlp(command, command, 0);
  while(wait(&status) != pid)
   continue;
  printf("\n");
 }
}
```

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);             /* syntax error if argc is not 3 */
```

```
/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);     /* open the source file */
if (in_fd < 0) exit(2);                          /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
if (out_fd < 0) exit(3);                         /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
if (rd_count <= 0) break;                        /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count); /* write data */
        if (wt_count <= 0) exit(4);              /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0)                               /* no error on last read */
        exit(0);
else
        exit(5);                                 /* error on last read */
}
```

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
        /* This function prints the thread's identifier and then exits. */
        printf("Hello World. Greetings from thread %d0, tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        /* The main program creates 10 threads and then exits. */
        pthread_t threads[NUMBER_OF_THREADS];
        int status, i;

        for(i=0; i < NUMBER_OF_THREADS; i++) {
                printf("Main here. Creating thread %d0, i);
                status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

                if (status != 0) {
                        printf("Oops. pthread_create returned error code %d0, status);
                        exit(-1);
                }
        }
        exit(NULL);
}
```

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

char * buf = "abcdefghijklmnopqrstuvwxyz";
int num_pthreads = 4;
int count = 60;
int fd = 1;

void * new_thread(void * arg)
{
 int i;
 for (i = 0; i < count; i++) {
   write(fd, arg, 1);
   sleep(1);
 }

 return(NULL);
 //pthread_exit(NULL);

}
```

```
main()
{
  pthread_t thread;
  int i;

  for (i = 0; i < num_pthreads; i++) {
   if (pthread_create(&thread, NULL,
                          new_thread, (void *)(buf + 3*i)))
   {
       fprintf(stderr, "error creating a new thread \n");
       exit(1);
   }
   pthread_detach(thread);
  }
  pthread_exit(NULL);
  return 0;
}
```

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS   3

void *BusyWork(void *null)
{
   int i;
   double result=0.0;
   for (i=0; i<1000000; i++)
   {
     result = result + (double)random();
   }
   printf("Thread result = %e\n",result);
   pthread_exit((void *) 0);
}

int main(int argc, char *argv[])
{
   pthread_t thread[NUM_THREADS];
   pthread_attr_t attr;
   int rc, t;
   void *status;

   /* Initialize and set thread detached attribute */
   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
```

```
for(t=0;t<NUM_THREADS;t++)
   {
      printf("Creating thread %d\n", t);
      rc = pthread_create(&thread[t], &attr, BusyWork, NULL);
      if (rc)
      {
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }

   /* Free attribute and wait for the other threads */
   pthread_attr_destroy(&attr);
   for(t=0;t<NUM_THREADS;t++)
   {
      rc = pthread_join(thread[t], &status);
      if (rc)
      {
         printf("ERROR return code from pthread_join() is %d\n", rc);
         exit(-1);
      }
      printf("Joined with thread %d status= %ld\n",t,(long)status);
   }

   pthread_exit(NULL);
}
```