Λειτουργικά Συστήματα

Πρώτη άσκηση

Τσάμπρας Κωνσταντίνος

up1083865

➔ 1) Υπολογισμός αθροίσματος με χρήση εργασιών.

Αρχικά θα χρησιμοποιήσουμε n διεργασίες για να υπολογίσουμε το άθροισμα των πρώτων N ακεραίων. Αυτό επιτυγχάνεται με το παρακάτω πρόγραμμα:

- Σύντομη περιγραφή:

Συμπεριλαμβάνουμε τις κατάλληλες βιβλιοθήκες (η unistd.h είναι για την fork()).

Δημιουργούμε την σωλήνωση (pipe) για την επικοινωνία της κύριας διεργασίας με τα παιδιά-διεργασίες.

Ελέγχουμε αν κλήθηκε σωστά το πρόγραμμα (αναμένουμε 3 ορίσματα).

Μετατρέπουμε τα ορίσματα σε ακεραίους.

processes_sum.c

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




int main(int argc, char *argv[]) {
    int fd[2];

    // create pipe
    pipe(fd);

    if (argc != 3) {
        printf("Usage: %s <N> <n>\n", argv[0]);
        return 1; // Exit with an error code
    }

    // Convert command line arguments to integers
    int N = atoi(argv[1]);
    int n = atoi(argv[2]);



    printf("Sum of the first %d integers, calculated by %d processes:\n", N, n);

```

Στη συνέχεια ξεκινάμε τον βρόχο επανάληψης για την δημιουργία των n διεργασιών.

Για κάθε διεργασία, απλά υπολογίζουμε το ι-οστό κομμάτι του αθροίσματος και το γράφουμε στο pipe. Έπειτα η κάθε διεργασία-παιδί κάνει exit.

```
26
27
28      for (int i=0; i < n; i++) {
29
30          int pid = fork();
31
32          if (pid == 0) { // child process
33
34              // calculate the sum of the i-th part
35              int sum = 0;
36              if(i==n-1){ // last child process has to calculate the rest of the sum
37                  for (int k = i*(N/n); k <=N; k++){
38                      sum +=k;
39                  }
40              }
41              else{ // other child processes calculate the sum of the i-th part
42                  for (int k = i*(N/n); k <(i+1)*(N/n); k++){
43                      sum +=k;
44                  }
45              }
46              // i-th child process writes the sum in the pipe and exits
47              close(fd[0]); // write, close read
48              write(fd[1], &sum, sizeof(sum));
49              close(fd[1]); // close write
50              exit(0); // exit child process
51          }
52      }
53
```

Τέλος υπολογίζουμε στο main process το άθροισμα με τον τύπο της αριθμητικής προόδου, αθροίζουμε όλα τα αποτελέσματα των διεργασιών και τυπώνουμε τα δύο αποτελέσματα.

```
53
54      // parent process calculates the actual sum
55      int parent_sum = N*(N+1)/2;
56
57
58      close(fd[1]); // read, close write
59
60      int children_sum = 0; // the sum of the sums of all the child processes
61      int val;
62      for (int i=0; i < n; i++) {
63          read(fd[0], &val, sizeof(val)); // waiting
64          printf("Parent receives value %d from child %d\n", val, i);
65          children_sum += val;
66      }
67
68      printf("The total child_sum is %d\n", children_sum);
69      printf("Computed parent_sum by parent process is %d\n", parent_sum);
70      close(fd[0]); // close read
71      return 0;
72  }
73
```

- Δοκιμές:
  - Α)

```
ultron@debian:~/ask1$ ./processes_sum
Usage: ./processes_sum <N> <n>
ultron@debian:~/ask1$ ./processes_sum 100 10
Sum of the first 100 integers, calculated by 10 processes:
Parent receives value 645 from child 0
Parent receives value 745 from child 1
Parent receives value 545 from child 2
Parent receives value 845 from child 3
Parent receives value 445 from child 4
Parent receives value 1045 from child 5
Parent receives value 345 from child 6
Parent receives value 245 from child 7
Parent receives value 145 from child 8
Parent receives value 45 from child 9
The total child_sum is 5050
Computed parent_sum by parent process is 5050
```

  - Β)

```
ultron@debian:~/ask1$ ./processes_sum 100 1
Sum of the first 100 integers, calculated by 1 processes:
Parent receives value 5050 from child 0
The total child_sum is 5050
Computed parent_sum by parent process is 5050
ultron@debian:~/ask1$
```

  - Γ)

```
ultron@debian:~/ask1$ ./processes_sum 100 100
Sum of the first 100 integers, calculated by 100 processes:
Parent receives value 6 from child 0
Parent receives value 7 from child 1
Parent receives value 5 from child 2
...

Parent receives value 199 from child 99
The total child_sum is 5050
Computed parent_sum by parent process is 5050
ultron@debian:~/ask1$
```

- Δ)

```
ultron@debian:~/ask1$ ./processes_sum 10000 100
Sum of the first 10000 integers, calculated by 100 processes:
Parent receives value 64950 from child 0
...
The total child_sum is 50005000
Computed parent_sum by parent process is 50005000
ultron@debian:~/ask1$
```

➔ 2) Υπολογισμός αθροίσματος με χρήση νημάτων.

Αυτήν την φορά θα χρησιμοποιήσουμε n νήματα γγια να υπολογίσουμε το άθροισμα των πρώτων N ακέραιων αριθμών.

- Σύντομη περιγραφή κώδικα

Ορίζουμε τις απαραίτητες global μεταβλητές στις οποίες θα έχουν πρόσβαση όλα τα νήματα, αυτές είναι το πλήθος των νημάτων, το N και το integer array που θα περιέχει το αποτέλεσμα του αθροίσματος του κάθε νήματος.

Το κάθε νήμα θα έχει ένα μοναδικό αναγνωριστικό (tid ή rank). Τα ranks ξεκινάνε από το 0 και αυξάνονται κατά ένα για κάθε νέο νήμα (σύμφωνα με την υλοποίηση μας παρακάτω), έτσι δεν χρειάζεται να κάνουμε κάτι σύνθετο για την επικοινωνία της διεργασίας με τα νήματα αφού το κάθε νήμα μπορεί να αποθηκεύει το αποτέλεσμα του υπολογισμού του στην αντίστοιχη θέση του array (arr[tid]).

Στην συνέχεια έχουμε την υλοποίηση της συνάρτησης που θα εκτελέσει το κάθε νήμα (work()). Η βιβλιοθήκη pthread απαιτεί η συνάρτηση προς εκτέλεση του κάθε νήματος να ακολουθεί το πρότυπο void * xxx(void *), οπότε ακολουθούμε αυτό το πρότυπο και περνάμε ως όρισμα το rank της μεταβλητής, αμέσως μετά το κάνουμε cast σε long για να μπορούμε να το χρησιμοποίησουμε. Έπειτα έχουμε τον γνωστό βρόγχο υπολογισμού του αθροίσματος του ι-οστού μέρους του ολικού αθροίσματος με την γνωστή εξαίρεση για το τελευταίο κομμάτι. Τέλος αποθηκεύουμε την τιμή του αθροίσματος στην θέση του array όπως προαναφέραμε.

threads.c

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




// global variables: accessible to all threads
int thread_count;
int N;
int *arr;


// thread function
void *work(void* rank) {
    long tid = (long) rank;

    // calculate the sum of the i-th part
    int thread_sum = 0;
    if(tid==thread_count-1){ // last child thread has to calculate the rest of the sum
        for (int k = tid*(N/thread_count); k <=N; k++){
            thread_sum +=k;
        }
    }
    else{ // other child thread calculate the sum of the i-th part
        for (int k = tid*(N/thread_count); k <(tid+1)*(N/thread_count); k++){
            thread_sum +=k;
        }

    }
    arr[tid]=thread_sum;

    return NULL;
}

```

Τέλος έχουμε την main.
Αρχικά παίρνουμε τα ορίσματα από το command line.
Δημιουργούμε όσα thread_handles (σύμφωνα με το πρότυπο) όσα το όρισμα και τα αποθηκεύομε σε ένα array για να τα χρησιμοποιήσουμε στη συνέχεια.
Έπειτα δεσμεύουμε τον χώρο μνήμης για το array των αποτελεσμάτων και δημιουργούμε τα νήματα καλώντας την pthread_create και παίρνοντας τα κατάλληλα ορίσματα σύμφωνα με το πρότυπο και με τις ανάγκες μας. Κάνουμε join ώστε η main διεργασία να περιμένει να τελειώσει το κάθε νήμα πριν φτάσει στο τέλος της η ίδια.
Τέλος υπολογίζουμε το άθροισμα από τον τύπο και προσθέτουμε τα επιμέρους αθροίσματα των νημάτων και τυπώνουμε τα αποτελέσματα.

```c

int main(int argc, char* argv[]) {
    long t;
    pthread_t* thread_handles;

    if (argc != 3) {
        printf("Usage: %s <N> <n>\n", argv[0]);
        return 1; // Exit with an error code
    }

    // Convert command line arguments to integers
    N = atoi(argv[1]);
    thread_count= atoi(argv[2]);

    printf("Sum of the first %d integers, calculated by %d threads:\n", N, thread_count);


    thread_handles = malloc (thread_count * sizeof(pthread_t));

    // global array
    arr = malloc (thread_count * sizeof(int));

    // create thread_count independent threads
    for (t = 0; t < thread_count; t++){
        pthread_create(&thread_handles[t], NULL, work, (void*) t);
    }

    printf("---Hello from the main thread---\n");

    // wait until threads are completed
    for (t = 0; t < thread_count; t++) {
        pthread_join(thread_handles[t], NULL);
    }

    // calculate the actual sum of the first N integers
    int process_sum = N*(N+1)/2;


    // sum of the sums of all the threads
    int sum=0;
    for (t = 0; t < thread_count; t++) {
        sum+=arr[t];
        printf("\tthread %ld returns %d\n" ,t ,arr[t]);
    }
    printf("Sum of the first %d integers (using threads) = %d\n", N, sum);
    printf("The process calculated %d\n", process_sum);
    free(thread_handles);
    free(arr);
    return 0;
}
```

- Δοκιμές:
  - Α)

```
ultron@debian:~/ask1$ gcc threads.c -o threads
ultron@debian:~/ask1$ ./threads
Usage: ./threads <N> <n>
ultron@debian:~/ask1$ ./threads 100 10
Sum of the first 100 integers, calculated by 10 threads:
---Hello from the main thread---
        thread 0 returns 45
        thread 1 returns 145
        thread 2 returns 245
        thread 3 returns 345
        thread 4 returns 445
        thread 5 returns 545
        thread 6 returns 645
        thread 7 returns 745
        thread 8 returns 845
        thread 9 returns 1045
Sum of the first 100 integers (using threads) = 5050
The process calculated 5050
ultron@debian:~/ask1$
```

- Β)

```
ultron@debian:~/ask1$ ./threads 100 1
Sum of the first 100 integers, calculated by 1 threads:
---Hello from the main thread---
        thread 0 returns 5050
Sum of the first 100 integers (using threads) = 5050
The process calculated 5050
ultron@debian:~/ask1$
```

- Γ)

```
ultron@debian:~/ask1$ ./threads 100 100
Sum of the first 100 integers, calculated by 100 threads:
---Hello from the main thread---
        thread 0 returns 0
        thread 1 returns 1
        thread 2 returns 2
        thread 3 returns 3
```

...

```
        thread 98 returns 98
        thread 99 returns 199
Sum of the first 100 integers (using threads) = 5050
The process calculated 5050
ultron@debian:~/ask1$
```

- Δ)

```
ultron@debian:~/ask1$ ./threads 10000 100
Sum of the first 10000 integers, calculated by 100 threads:
---Hello from the main thread---
        thread 0 returns 4950
        thread 1 returns 14950
        thread 2 returns 24950
```

...

```
        thread 98 returns 984950
        thread 99 returns 1004950
Sum of the first 10000 integers (using threads) = 50005000
The process calculated 50005000
ultron@debian:~/ask1$
```

➔ Σημείωση: Στις παραπάνω υλοποιήσεις τα αποτελέσματα του κάθε νήματος/διεργασίας τυπώνονται με την σειρά (και την αρίθμηση) με την οποία «συλλέγονται» από την main. Στην περίπτωση των διεργασιών η σειρά αυτή είναι τυχαία εφόσον η σειρά με την οποία οι διεργασίες είναι και αυτή τυχαία (ή τέλος πάντων μη προβλέψιμη από οποιονδήποτε εκτός του λειτουργικού). Αντιθέτως τα νήματα (αν και αυτά εκτελούνται και τελειώνουν με τυχαία σειρά) αποθηκεύουν τα

αποτελέσματα τους με σειρά, δηλαδή το νήμα νούμερο 0 θα αποθηκεύσει το αποτέλεσμα του στην θέση 0 του arr, το νήμα 1 στην θέση 1 κτλ, ανεξάρτητα από το πότε θα τελειώσει την εκτέλεσή του. Άρα και η main θα διαβάσει τα αποτελέσματα με «σειρά» αφού έχουν τελειώσει όλα τα νήματα.
Για αυτόν τον λόγο βλέπουμε να τυπώνονται με τυχαία σειρά τα επιμέρους αποτελέσματα των διεργασιών, ενώ στα νήματα βλέπουμε την «σωστή σειρά».