Λειτουργικά Συστήματα Άσκηση 1

2024-2025

AM: 1096060

Ονοματεπώνυμο: Μαρία-Νίκη Ζωγράφου

Σύντομη περιγραφή της λύσης για threads:

1. Φτιάχνουμε ειδικό `struct thread_data` για να μπορούμε να περάσουμε πολλά ορίσματα σε συνάρτηση που θα τρέχει σε πολλά threads ταυτόχρονα. Τα ορίσματα που θα χρειαστούμε είναι: [1. Από πού θα αρχίσει το άθροισμα 2.Που θα σταματήσει. 3.Χώρος για αποθήκευση της μεταβλητής αποτελέσματος.]

```c
/**pthread_create function allows specific arguement (επιτρέπονται μόνο τα εξής ορίσματα:)*/
                                    /*int pthread_create(pthread_t * thread, const pthread_attr_t * attr, */
typedef struct thread_data          /* void * (*start_routine)(void *), void *arg); */
                                    /**therefore multiple arguements can be given to the thread only with a struct (+we cant return int)*/
                                    /**but we can give an int result as part of the struct*/
{                                   /**άρα πολλαπλά ορίσματα μόνο με struct γίνεται να περαστούν*/
    int  start;     /*αρχή σειράς αθροίσματος*/
    int  end;       /*τέλος σειράς αθροίσματος*/
    int result;     /*αποτέλεσμα*/
} Thread_data;
```

2.Στη main διαχειριζόμαστε το input του χρήστη στη main και το περνάμε ως όρισμα σε μια συνάρτηση που αναλαμβάνει το threading(`tot=threader(N,n);`). Στο τέλος συγκρίνουμε τον υπολογιμσό της συνάρτησης με τον τύπο N*(N+1)/2.

```c
int main()
{
    int escape=0;
    int N;
    int n;
    int tot;
    while(escape==0){                                   /**< input handling */
        printf("Type number N: \n");
        scanf("%d", &N);
        printf("Type number n: \n");
        scanf("%d", &n);
        if(N%n!=0){
            printf("N should be a multiple of n!\n");
            continue;
        }
        tot=threader(N,n);                              /**calls function where threading happens*/
        printf("calculation=%d\n",N*(N+1)/2);
        if(tot!=N*(N+1)/2){printf("mistake\n");}        /*check*/
        printf("if you want to repeat press zero: '0'\n");
        scanf("%d", &escape);
    }

    return 0;
}
```

3. Χρησιμοποιείται η συνάρτηση `int threader(int N, int n),` που δημιουργεί n threads. Για κάθε thread δημιουργείται και ένα αντίστοιχο

Thread_data με τα κατάλληλα αρχικοποιημένα δεδομένα. Τα αποτελέσματα της πρόσθεσης φυλάγονται στα struct thread_data τα οποία βρίσκονται σε μια λίστα.

Αναλυτικότερα δημιουργούμε ένα array με στοιχεία pthread_t, που είναι ουσιαστικά ακέραιοι αριθμοί που αντιστοιχεί ο καθένας σε ένα νήμα. Για κάθε νήμα που δημιουργούμε, θα φυλάσσεται εκεί ο αριθμός του. Δημιουργούμε επίσης ένα array με στοιχεία δείκτες (pointers) σε Thread_data. Για κάθε νήμα δημιουργούμε ένα νέο Thread_data και το αρχικοποιούμε κατάλληλα. Ύστερα το περνάμε σαν argument στο αντίστοιχο thread.
Αφού δημιουργήσουμε όλα τα thread, κάνουμε `pthread_join` για όλα ώστε να εξασφαλίσουμε ότι θα τρέξουν και θα τερματίσουν πρώτα όλα τα νήματα και ύστερα θα κληθούν άλλες συναρτήσεις και διεργασίες.

```
51  int threader(int N, int n){
52      int num_of_threads;int tot;int k=N/n;int i=0;
53      /*int pthread_create(pthread_t * thread,*/
54      pthread_t* pthread_t_arr; /*thread: pointer to an unsigned integer value that returns the thread id of the thread created. Geeksforgeeks*/
55      Thread_data** thread_data_arr; /**array with poinetrs to structs, so we initialize pointer to pointer to Thread_data*/
56      pthread_t_arr=malloc(n*sizeof(pthread_t)); /*array of pointers to threads' ids*/
57      thread_data_arr=malloc((n+1)* (sizeof (Dataptr )));
58      thread_data_arr[n]=NULL; /*used in while loop in total == symbolises end of list*/
59
60      for(num_of_threads=0; num_of_threads<n; num_of_threads++) /*every iteration=new thread*/
61      {
62          thread_data_arr[num_of_threads]=malloc(sizeof(Thread_data)); /*struct thread data for each thread initialized*/
63          if(thread_data_arr[num_of_threads]==NULL)
64          {
65              printf("null data");
66              exit(1);
67          }
68          thread_data_arr[num_of_threads]->start=k*num_of_threads+1;/**< 1 to k, k+1 to 2k etc: [from iteration*k+1 up to (iteration+1)*k] */
69          thread_data_arr[num_of_threads]->end=k*(num_of_threads+1);
70          thread_data_arr[num_of_threads]->result=0; /**\initiallization \for \safety*/
71  /**< new thread created with the adder as a subroutine and the struct thread_data as arguements*/
72          if (pthread_create(&pthread_t_arr[num_of_threads],NULL,adder,(void *)thread_data_arr[num_of_threads])!= 0)
73          {
74              printf("Error creating thread\n"); /*if it fails*/
75              exit(1);
76          }
77
78      }/**pthread_join all threads so they are all required to finish before total is called and main terminates*/
79      for (num_of_threads = 0; num_of_threads < n; num_of_threads++)
80      {
81          pthread_join(pthread_t_arr[num_of_threads], NULL);
82
83      }
84      tot=total(thread_data_arr); /*total is called to add all the partial results stored in the thread_data_arr*/
85      printf("total result=%d\n",tot); /**< threads communicate through common memory!!! */
86      return tot;
87  }
```

4. Η συνάρτηση `void* adder(void *arg)` καλείται. Προσθέτει όλους τους ενδιάμεσους ακέραιους αριθμούς από έναν αρχικό έως έναν τελευταίο. Έχει αυτό το πρότυπο ώστε να μπορεί να περαστεί σαν όρισμα στην: `int pthread_create(pthread_t* thread, const pthread_attr_t* attr, void * (*start_routine)(void *), void *arg);`. Οι αριθμοί έχουν χωριστεί σε n μέρη άρα πρέπει κάθε φορά 'i' να προσθέτει από τον i*k+1 έως τον (i+1)*k, όπου k=N/n.

```
103 void* adder(void *arg)  /**start from start and stop at end. add all the numbers in between. e.g. 6+7+8+9+10
104 {
105     int i=0;
106     struct thread_data *data;
107
108     data = (struct thread_data *) arg;
109     /*printf("thread created data %p\n",data);*/
110     for(i=data->start; i<=data->end; i++)
111     {
112         /*printf("%d\n",i);*/
113         data->result+=i;
114     }
115     /*printf("result %d\n",data->result);*/
116     return NULL;
117 }
118
```

5. Η συνάρτηση `int total(Thread_data** data_arr)` καλείται από την threader και διατρέχει τη λίστα με στοιχεία τις δομές δεδομένων Thread_data. Αυτή η λίστα αποτελεί κοινή μνήμη των διεργασιών. Χάρη στην κοινή μνήμη επιτυγχάνεται επικοινωνία ανάμεσα στα νήματα και όλα τα επιμέρους αποτελέσματα αθροίζονται σε ένα συνολικό.

```
 90  int total(Thread_data** data_arr)
 91  {
 92      int i=0;
 93      int tot=0;
 94      if (data_arr==NULL)return -1;
 95      while(data_arr[i]!=NULL)
 96      {
 97          tot+=data_arr[i]->result;
 98          i++;
 99      }
100      return tot;
101
```

Σύντομη περιγραφή της λύσης για processes:

/*Υπάρχουν και αναλυτικά σχόλια στο αρχείο κώδικα*/

Εξήγηση συναρτήσεων:

```
int *inputs(void);
int checker(unsigned long long int calculation,unsigned long long int
N);
int multiprocessing(int N,int n);
unsigned long long int totalcalc(int p[],int n);
unsigned long long int calculator(int start,int end);
void child(int p[],int N,int n,int i);
int main()
```

main():

- Βήμα 1: Με την `int *inputs(void)` διαχειριζόμαστε το input του χρήστη επιστρέφοντας N,n.
- Βήμα 2: Καλούμε την συνάρτηση `int multiprocessing(int N, int n);` η οποία αναλαμβάνει την δημιουργία n διεργασιών, τον υπολογισμό και έλεγχο του αθροίσματος.
- Βήμα 3: Ελέγχουμε αν ο χρήσης θέλει να συνεχίσει η εκτέλεση του προγράμματος και αγνοούμε άκυρα inputs. Αν δοθεί '0' το πρόγραμμα συνεχίζει, αλλιώς τερματίζεται.

int multiprocessing(int N,int n):

- Βήμα 1:Δημιουργία pipe.
- Βήμα 2: Βρόχος για την δημιουργία n διεργασιών με `int fork()`.
- Βήμα 3: Ελέγχουμε αν πρόκειται για διεργασία γονέα, ώστε να μην κάνουμε τίποτα, ή παιδιού για να κληθεί η συνάρτηση `void child(int p[],int N,int n,int i);` . Ελέγχουμε επίσης αν το fork εκτελέσθηκε επιτυχώς.
- Βήμα 4: Κλείνουμε το Pipe και περιμένουμε να τελειώσουν όλα τα παιδία, καλώντας την wait n φορές. Η `Wait()` θα περιμένει μέχρι να τελειώσει ένα οποιοδήποτε παιδί του Parent, εμείς όμως έχουμε n παιδιά.
- Βήμα 5: Καλούμε την `unsigned long long int totalcalc(int p[],int n);` που υπολογίζει το συνολικό άθροισμα.
- Βήμα 6: Καλούμε τον `int checker(unsigned long long int calculation,unsigned long long int N);`που ελέγχει την ορθότητα του αποτελέσματος.

unsigned long long int totalcalc(int p[],int n):

- Βήμα 1: Βρόχος n επαναλήψεων.
- Βήμα 2: Ανάγνωση από το pipe τα αποθηκευμένα αθροίσματα και υπολογισμός συνόλου.
- Βήμα 3: Εκτύπωση και επιστροφή αποτελέσματος.

int *inputs(void):

- Βήμα 1: Ανάγνωση N και n από χρήστη.
- Βήμα 2: Επιστροφή σε array.

int checker(unsigned long long int calculation, unsigned long long int N):

- Βήμα 1: Χρήση τύπου για σύνολο και σύγκριση με υπολογισμό της `totalcalc`.
- Βήμα 2: Επιστροφή 1 (σωστό) ή 0 (λάθος).

unsigned long long int calculator(int start,int end):

- Βήμα 1:Βρόχος από start έως end και άθροισμα όλων ενδιάμεσων ακεραίων.
- Βήμα 2: Επιστροφή αθροίσματος.

void child(int p[],int N,int n,int i):

- Βήμα 1:Καλούμε την calculator με τα κατάλληλα ορίσματα. Η αρχή της θα είναι i*N/n, καθώς η child() καλείται μέσα στον βρόχο των n processes και πρέπει κάθε φορά να προσθέσει N/n αριθμούς. Το δεύτερο όρισμα του τέλους θα είναι N+1, αν είμαστε στην n-1 επανάληψη ώστε να προστεθεί και ο αριθμός N στο άθροισμα. Αλλιώς θα είναι (i + 1) * (N / n) για να μην πορστεθει ο αριθμος (i + 1) * (N / n) δύο φορές (θα συμπεριληφθεί για το επόμενο i ως start (i*N/n)).

```
sum = calculator(i*(N / n),(i < n - 1) ? ((i + 1) * (N / n)):(N + 1));
```

- Βήμα 2: Γράφουμε στο pipe το τοπικό άθροισμα που υπολογίσαμε.

Σημείωση:

Εναλλακτικός τρόπος υπολογισμού αθροίσματος, σε πιο συμπτυγμένο αλλά και πιο δυσανάγνωστο κώδικα:

```
for(int j=i*k+1;j<=(i+1)*k;j++){
      //range of numbers to be added: form i*k+1 up to (i+1)*k
      sum+=j;
}
```

Συμπέρασμα για τρόπο επικοινωνίας:

Στο threading τα νήματα μοιράζονται κοινή μνήμη, οπότε η επικοινωνία μεταξύ των διαφορετικών νημάτων γίνεται χάρη σε αυτή την κοινή μνήμη. Συγκεκριμένα χρησιμοποιήθηκε λίστα με Pointers σε Thread_data στοιχεία. Στο multiprocessing οι διεργασίες δεν έχουν κοινή μνήμη αλλά επικοινωνούν με pipes. Στο pipe μπορούμε να γράψουμε από την μία άκρη και διαβάζουμε από την άλλη. Κάθε παιδί γράφει στο pipe τους υπολογισμούς του και ο γονέας στο τέλος διαβάζει ότι έγραψαν τα παιδιά. (το pipe λειτουργεί παρόμοια με μία ουρά queue).