**LAB 7. Thread-uri**

7.2

Intr-un proces putem avea mai multe fire de executie ce ruleaza in paralel. SO ne permite sa permite sa pornim un thread nou sau sa asteptam terminarea unuia.

**7.2.1 creare thread. asteptare terminare thread**

**| #include <pthread.h>**

**| int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,void \*(\*start\_routine) (void \*), void \*arg);**

**| int pthread\_join(pthread\_t thread, void \*\*retval);**

OBS: pe langa pthread.h, e nevoie sa includem si biblioteca pthread in etapa de link-aare a programului.

--> se adauga la apelul compilatorului optiunea -pthread ==> marcheaza in executabil faptul ca se foloseste cod din aceasta biblioteca si cere so sa o incarce in executa programului.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|**gcc -Wall program.c -o program -pthread** |

------------------------------------------

**7.2.1.1 pthread\_create**

**| int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,void \*(\*start\_routine) (void \*), void \*arg);**

--> permite pornirea unui nou fir de executie ce va executa codul functiei **start\_routine**, transmisa prin al treilea argument

-> explicatii argumente:

🡪 functia executata de threat trebuie sa primeasca un parametru de tip void\* si rezultatul rezultat sa fie void\*

🡪 **primul argument (\*thread)** --> adresa unei variabile in care functia va completa **id-ul thread-ului creat in caz de succes**

🡪 **al doilea argument (\*attr)** --> pointer in care putem specifica **atributele** **thread**-ului; daca dorim sa folosit atributele implicite ==> pointerul va fi NULL

**🡪 argumentul al treilea argument (\*start\_routine)** --> **pointer la functia ce va fi executata de thread**

🡪 argumentul al patrulea (\*arg) --> argument generic \*void --> poate fi adresa oricarei structuri

* in caz de succes, pthread\_create(..) returneaza 0;

**7.2.1.2 pthread\_join**

**| int pthread\_join(pthread\_t thread, void \*\*retval)**

**🡺 ne permite sa ateptam terminarea executiei unui thread.**

🡪 **primul argument (thread) --> tid** 🡪 obtinut in urma apelului functiei pthread\_create

🡪 **retval** 🡺 poiter modificat a.i. sa indice spre rezultatul functei thread-ului

🡺 pointer la un pointer de tip void\* deoarece functia thread-ului rezunteaza un rezultat de top void\* ( adresa unei vb/structuri alocate dinamic in care sunt stocate rezultatele functei)

**7.2.1.3 Exemplu**

**1 void \*threadFn(void \*unused)**

**2 {**

**3 printf("Secondary thread\n");**

**4 return NULL;**

**5 }**

**6**

**7 int main()**

**8 {**

**9 pthread\_t tid = -1;**

**10**

**11 if(pthread\_create(&tid, NULL, threadFn, NULL) != 0){**

**12 perror("Error creating thread");**

**13 return 1;**

**14 }**

**15 printf("Main thread\n");**

**16 pthread\_join(tid, NULL);**

**17 return 0;**

**18 }**

🡪 in functia principala se apeleaza pthread\_create pentru a porni thread-ului.

🡪 tid = identificatorul thread-ului; folosit ulterior la asteptarea terminarii unui thread

**🡪 firul principal de executie merge de la inceputul pana la sfarsitul programului.**

**🡪 firul secundar incepe de la crearea thread-ului (linia 11) si continua pana la terminarea executiei functiei thread-ului**

Diagram

Description automatically generated **🡺 obs: terminarea executiei nu trebuie neaparatsa coincida cu apelul pthread\_join pe thread-ul principal**

🡺 **SCENARII:**

**-> functia thread-ului isi incheie executia inainte sa se apeleze pthread\_join ( join-ul rezultat va fi disponibil imediat si thread-ul principal poate continua)**

**-> funtia thread-ului isi incheie executia dupa ce se apeleaza pthread\_join (thread-ul principal este blocat, intrerupandu-si executia pana la inchiderea functiei thread-ului)**

**7.2.2 transmiterea argumentelor unui thread**

--> o functie de thread va avea tot timpul **prototipul void \*threadFn(void \*)** 🡺 primeste mereu argument de tip void\* si returneaza rezultate de tip void\*

🡺 util la flexibilitate; pointerul de tip void\* poate converti in orice alt tip de pointer, inclusiv catre o structura cu cate campuri avem nevoie

7.2.2.1 exemplu --> cerinta: maximul dintr-un vector random cu tread-uri; impartim vectorul in bucati de dimensiuni aprox egale si fiecare thread calculeaza maximul dintr-o anumita bucata

**7.2.2.2**

**1 typedef struct {**

**2 int \*arr;**

**3 int from;**

**4 int to;**

**5 } TH\_STRUCT;**

**6**

**7 void \*thread\_fn\_max(void \*param)**

**8 {**

**9 TH\_STRUCT \*s = (TH\_STRUCT\*)param; // conversie la structura a.i. sa poata fi deferentiat si sa se poate extrage info**

**10 int i, max;**

**11 max = s->arr[s->from];**

**12 for(i=s->from; i<=s->to; i++){**

**13 if(s->arr[i] > max){**

**14 max = s->arr[i];**

**15 }**

**16 }**

**17 return (void\*)(long)max; // cast dublu --> daca am fi transformat direct din int in void\*, compilatorul ar fi afisat un warning deoarece pe 64bit un intreg si un pointer au dim diferite(4 si 8 octeti)**

**18 }**

explicatii: --> structura e necesara pentru a contine toate informatile necesare

--> functia rezurneaza tipul void\* --> poate ocupa 8 octeti (pe 64bit) sau 4 octeti (pe 32bit)

==> e necesar sa facem cast

**7.2.2.3 Generarea vectorului aleator**

**1 #define SIZE\_ARRAY 1000**

**2**

**3 void generate\_random\_array(int \*v, int size)**

**4 {**

**5 int i;**

**6 for(i=0; i<size; i++){**

**7 v[i] = rand() % 1000000;**

**8 }**

**9 }**

**10**

**11 int main()**

**12 {**

**13 int v[SIZE\_ARRAY];**

**14 ...**

**15**

**16 srand(time(NULL));**

**17 generate\_random\_array(v, SIZE\_ARRAY);**

**18 ...**

**19 }**

**7.2.2.4 impartirea vectorlui intre thread-uri**

**1 #define SIZE\_ARRAY 1000**

**2 #define NR\_THREADS 6**

**3**

**4 int main()**

**5 {**

**6 ...**

**7 TH\_STRUCT params[NR\_THREADS];**

**8 int i;**

**9 ...**

**10**

**11 for(i=0; i<NR\_THREADS; i++){**

**12 params[i].arr = v;**

**13 if(i == 0){**

**14 params[i].from = 0;**

**15 }else{**

**16 params[i].from = params[i-1].to + 1;**

**17 }**

**18 params[i].to = params[i].from + SIZE\_ARRAY / NR\_THREADS - 1;**

**19 if(i < SIZE\_ARRAY % NR\_THREADS){**

**20 params[i].to++;**

**21 }**

**22**

**23 ... //create thread**

**24 }**

**25 ...**

**26 }**

// explicatii

* impartim munca intre thread-uri cat mai echitabil ; incercam sa impartim vectorul in SIZE\_ARRAY/NB\_THREADS elemente procesate de fiecare thread ;
  + - daca impartirea nu e exacta 🡺 **SIZE\_ARRAY % NR\_THREADS** thread-uri vor procesa cate un element in plus

**7.2.2.5 lansarea si asteptarea thread-urilor**

**1 int main()**

**2 {**

**3 pthread\_t tid[NR\_THREADS];**

**4 TH\_STRUCT params[NR\_THREADS];** // pastram tid-ul fiecarui thread pentru

**5 int i;**

**6 void \*result;**

**7 int thMax, max = 0;**

**8 ...**

**9**

**10 for(i=0; i<NR\_THREADS; i++){**

**11 ... // fill params[i]**

**12 pthread\_create(&tid[i], NULL, thread\_fn\_max, &params[i]);**

**13 }**

**14 for(i=0; i<NR\_THREADS; i++){**

**15 pthread\_join(tid[i], &result); //** asteptam terminarea thread-urilor 🡪 rezultatul e de tip void\*🡪 trebuie sa folosim pointer la o variabila de acest tip

**16 thMax = (int)(long)result**; // conversie pointer la intreg

**17 if(thMax > max){**

**18 max = thMax;**

**19 }**

**20 }**

**21 printf("The maximum value is %d.\n", max);**

**22**

**23 return 0;**

**24 }**

// explicatii

* cand transmitem info diferite thread-urilor, e important ca fiecare thread sa aiba parametrii intr-o structura diferita,decoarece thread-urile primesc doar o referinta la structura, NU O COPIE
* asteptarea thread-urilor trebuie facuta in bucla separata fata de cea de creeare; saca asteptam dupa terminarea thread-ului imediat dupa ce a fost creat, se obtine o performanta similara cu apelul direct al functiilor, fara sa se profite de paralelism

**7.2.3 inchiderea executiei unui thread**

**#include <pthread.h>**

**void pthread\_exit(void \*retval);**

**int pthread\_cancel(pthread\_t thread);**

**int pthread\_setcancelstate(int state, int \*oldstate);**

**int pthread\_setcanceltype(int type, int \*oldtype);**

**7.2.3.1 pthread\_exit**

void pthread\_exit(void \*retval);

**--> un thread isi incheie executia atunci cand functia acestuia se termina, sau prin apelul pthread\_exit**

--> retval = valoarea returnata de thread

**7.2.3.2 pthread\_cancel**

int pthread\_cancel(pthread\_t thread);

--> un thread poate fi anulat/inchis si de a**lt thread din acelasi proces**

--> nu se recomanda inchiderea fortata a unui thread si se prefera ca thread-urile sa isi incheie singure executia;

--> atunci cand un thread e inchis din exterior, acesta poate lasa resurse folosite in stare inconsistenta ( fisiere deschise si ineinchise, vector completat partial..)

--> cand se apeleaza **pthread\_cancel** , **thread-ul tinta nu se va inchide istant , ci doar in momentul in care acesta atunge un punct de terminare (cancellation point),** in anumite apeluri de sistem(open,close,read,write,sleep)

**7.2.3.3 pthread\_setcancelstate si pthread\_setcanceltype**

**int pthread\_setcancelstate(int state, int \*oldstate);**

**int pthread\_setcanceltype(int type, int \*oldtype);**

--> in mod implicit, un thread poate fi anulat de thread-uri din acelasi proces

**🡪 totusi, un thread poate sa isi modifice starea cu pthread\_setcancelstate**

==> parametrul state poate sa fie:

 **PTHREAD\_CANCEL\_ENABLE** - thread-ul curent devine anulabil (implicit);

 **PTHREAD\_CANCEL\_DISABLE** - thread-ul curent **nu** se va putea anula prin apelul **pthread\_cancel**

**🡪 implicit, daca un thread e anulabil, inchiderea acestuia se va face doar cand se ajunge la un cancellation point**

**🡪** se poate modifica acest tip de anulabilitate, cu functia **pthread\_setcanceltype**

==> parametrul type poate sa fie:

** PTHREAD\_CANCEL\_DEFERRED** - thread-ul curent se poate inchide doar cand atinge un punct de anulare (implicit);

** PTHREAD\_CANCEL\_ASYNCHRONOUS** - thread-ul curent se poate inchide imediat ce se apeleaza pthread\_cancel

**Intrebari**:

**7.3.1 ˆIntreb˘ari recapitulative**

**1. Cum se compileaz˘a ˆın linia de comand˘a programele care folosesc thread-uri?**

R: gcc -Wall program.c -o program -**pthread**

**2. Ce primes, te ca argument s, i ce returneaz˘a o funct, ie de thread?**

**🡪 create: primeste arumente de tip void\* , referinta la functia pe care o va executa, attr 🡪 pointer in care se specifica atributele; returneaza pointer de tip void\***

**3. Cum proced˘am atunci cˆand funct, ia de thread trebuie s˘a primeasc˘a mai multe argumente?**

**🡪 folosim un struct**

**4. Ce este gres, it din punct de vedere al logicii programului ˆın codul de mai jos?**

🡪 functia creeaza un thread, dupa care imediat asteapta terminarea lui 🡺 performanta similara cu apelul direct al unei functii si nu se poate profita de paralelismul oferit de thread-uri

**5. Cˆand ˆıs, i ˆıncheie execut, ia un thread pentru care s-a apelat pthread cancel?**

**🡪 cand atinge cancellation point**