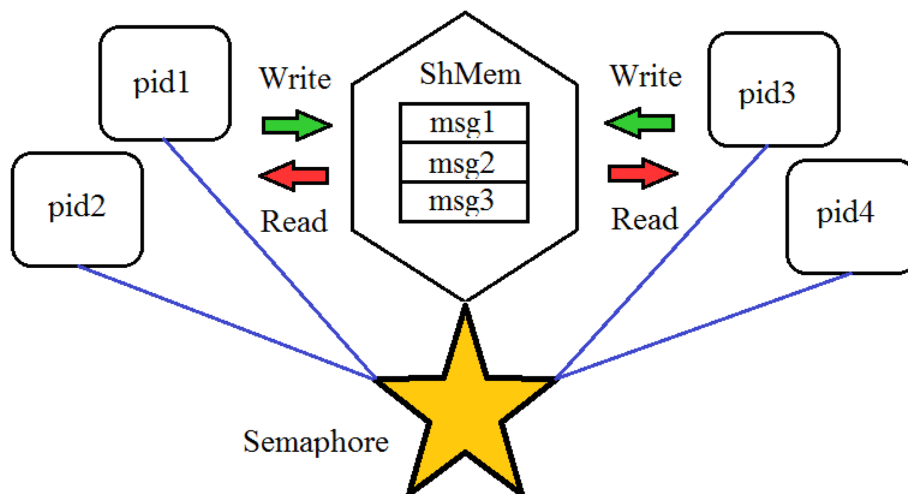
Семинар 3. Использование семафоров и общих сегментов памяти.

1. Другие способы обмена между локальными процессами.

1) разделяемые сегменты общей памяти (функции shmget - создание на сервере, shmat - присоединение, shmdt - отсоединение, shmctl - удаление);
2) семафоры как механизм доступа к общим переменным (функции semget, semop, semctl);
3) легкие семафоры – mutex.
4) передача большого числа параметров в трэды.



2. Стандарт OpenMP. Базовые конструкции.

Интерфейс OpenMP – стандарт для программирования на масштабируемых SMP-системах (SSMP, ccNUMA и других) в модели общей памяти (*shared memory model*). В стандарт OpenMP входят спецификации набора директив компилятора, вспомогательных функций и переменных среды. OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» (master) поток создает набор «подчиненных» (slave) потоков, и задача распределяется между ними. Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами, причём количество процессоров не обязательно должно быть больше или равно количеству потоков.
POSIX-интерфейс для организации нитей (Pthreads) поддерживается во всех ОС. OpenMP можно рассматривать как высокоуровневую надстройку над Pthreads (или аналогичными библиотеками нитей); в OpenMP используется терминология и модель программирования, близкая к Pthreads. Например, динамически порождаемые нити, общие и разделяемые данные, механизм «замков» для синхронизации и т.д.. Согласно терминологии POSIX threads, любой процесс состоит из нескольких нитей управления, которые имеют общее адресное пространство, но разные потоки команд и раздельные стеки. В простейшем случае процесс состоит из одной нити (потока, легковесного процесса, LWP - light-weight processes).
Важным достоинством технологии OpenMP является возможность реализации так называемого ***инкрементального программирования***, когда программист постепенно находит участки в программе, содержащие ресурс параллелизма, с помощью предоставляемых механизмов делает их параллельными, а затем переходит к анализу следующих участков. Таким образом, в программе нераспараллеленная часть постепенно становится всё меньше.
На сайте книга Антонова: http://polyakov.imamod.ru/arc/stud/parallel/Antonov_OpenMP.pdf

Пример 1 (ex05a.c). Вычисление однократного интеграла, тяжелые процессы.
Пример 2 (ex05b.c). Вычисление однократного интеграла, легкие процессы.
Пример 3 (ex05c.c). Вычисление однократного интеграла с помощью OpenMP.

Тяжелые процессы:

```
#include <stdio.h> #include <stdlib.h>
#include <unistd.h> #include <math.h>
#include "mycom.h"
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#define SEM_ID 2007
#define SHM_ID 2008
#define PERMS  0600
typedef struct tag_msg_t
{ int n; double s; } msg_t;
#ifdef _SEM_SEMUN_UNDEFINED
union semun {
  int val;
  struct semid_ds *buf;
  unsigned short int *array;
  struct seminfo *__buf;
};
#endif
int np, mp;
int semid;
int shmid;
msg_t* msg;
struct sembuf sem_loc   = {0,-1,0};
struct sembuf sem_unloc = {0, 1,0};
union semun s_un;
double a = 0, b = 1;
int ni = 1000000000;
double sum = 0;
double f1(double x);
double f1(double x) {return 4.0/(1.0+x*x);}
void myjobp();
void myjobp()
{
  int n1; double a1, b1, h1, s1;
  n1 = ni / np; h1 = (b - a) / np;
  a1 = a + h1 * mp;
  if (mp<np-1) b1 = a1 + h1; else b1 = b;
  s1 = integrate(f1,a1,b1,n1);
  printf("mp=%d a1=%le b1=%le n1=%d
s1=%le\n",mp,a1,b1,n1,s1);
  while(semop(semid,&sem_loc,1)<0); // wait +
lock
  msg->n++; msg->s += s1;
  while(semop(semid,&sem_unloc,1)<0); // wait
+ unlock
  if (mp == 0){ n1 = 0;
    while(n1<np){
      while(semop(semid,&sem_loc,1)<0); //
wait + lock
      n1 = msg->n; sum = msg->s;
      while(semop(semid,&sem_unloc,1)<0); //
wait + unlock
    }
  }
  return;
}

void NetInit(int np, int* mp);
void NetInit(int np, int* mp)
{
```

Легкие процессы:

```
#include <stdio.h> #include <stdlib.h>
#include <unistd.h> #include <math.h>
#include "mycom.h"
#include <pthread.h>

typedef struct tag_data_t {
  int n, nt, mt; double a, b, s, *sum;
} data_t;

int nt, mt;
pthread_mutex_t mut =
PTHREAD_MUTEX_INITIALIZER;
pthread_t* threads;
data_t* data;

double a = 0, b = 1;
int ni = 1000000000;
double sum = 0;
double f1(double x);
double f1(double x) {return 4.0/(1.0+x*x);}
void* myjobt(void* d);
void* myjobt(void* d)
{
  int n1; double a1, b1, h1;
  data_t* dd = (data_t *)d;
  n1 = dd->n / dd->nt;
  h1 = (dd->b - dd->a) / dd->nt;
  a1 = dd->a + h1 * dd->mt;
  if (dd->mt<dd->nt-1) b1 = a1 + h1;
  else b1 = dd->b;
  dd->s = integrate(f1,a1,b1,n1);
  printf("mt=%d a1=%le b1=%le n1=%d
s1=%le\n",dd->mt,a1,b1,n1,dd->s);
  pthread_mutex_lock(&mut); // lock
  *dd->sum += dd->s;
  pthread_mutex_unlock(&mut); // unlock

  return 0;
}

void ThreadInit();
void ThreadInit()
{
```

```
  int i; pid_t spid = 0;
  if (np>1){
    *mp=1; spid = fork();
    if (spid > 0 && np > 2)
      for (i=2;i<np;i++)
        if (spid > 0){ *mp=i; spid = fork();}
    if (spid > 0) *mp = 0;
  }
  else *mp = 0;
  return;
}

void ShrMemInit();
void ShrMemInit()
{
  if (mp == 0) {
    if ((shmid =
shmget(SHM_ID,sizeof(msg_t),PERMS |
IPC_CREAT)) < 0)
      if ((shmid =
shmget(SHM_ID,sizeof(msg_t),PERMS)) < 0)
        myerr("Can not find shared memory
segment",1);
    if ((msg = (msg_t*) shmat(shmid, 0, 0))
== NULL)
      myerr("Can not attach shared memory
segment",2);
    msg->n = 0; msg->s = 0; // initialization
of shared memory
    if ((semid = semget(SEM_ID, 1, PERMS |
IPC_CREAT )) < 0)
      if ((semid = semget(SEM_ID, 1, PERMS))
< 0)
        myerr("Can not find semaphore",3);
    s_un.val = 1;
semctl(semid,0,SETVAL,s_un); // unlock
  }
  else {
    while((shmid =
shmget(SHM_ID,sizeof(msg_t),PERMS)) < 0);
    if ((msg = (msg_t*) shmat(shmid, 0, 0))
== NULL)
      myerr("Can not attach shared memory
segment",4);
    while((semid = semget(SEM_ID, 1, PERMS))
< 0);
  }
  return;
}

void ShrMemDone();
void ShrMemDone()
{
  if (mp == 0) {
    if (semctl(semid, 0, IPC_RMID, (struct
semid_ds *) 0) < 0)
      myerr("Can not remove semaphore",5);
    if (shmdt(msg)<0)
      myerr("Can not dettach shared memory
segment",6);
    if (shmctl(shmid, IPC_RMID, (struct
shmid_ds *) 0) < 0)
      myerr("Can not remove shared memory
segment",7);
  }
  else {
    if (shmdt(msg)<0)
      myerr("Can not dettach shared memory
```

```
  int i;
  if (!(threads = (pthread_t*)
malloc(nt*sizeof(pthread_t))))
    myerr("Not enough memory",1);
  if (!(data = (data_t*)
malloc(nt*sizeof(data_t))))
    myerr("Not enough memory",1);
  for (i=0; i<nt; i++){
    (data+i)->a=a;
    (data+i)->b=b;
    (data+i)->n=ni;
    (data+i)->nt=nt;
    (data+i)->mt=i;
    (data+i)->sum = &sum;
    if
(pthread_create(threads+i,0,myjobt,(void*)(
data+i)))
      myerr("Can not create thread",2);
  }
  return;
}

void ThreadDone();
void ThreadDone()
{
  int i;
  for (i=0; i<nt; i++)
    if (pthread_join(threads[i],0))
      myerr("Can not wait thread",3);
  free(data);
  free(threads);
  return;
}
```

```
segment",8);
    exit(0);
  }
  return;
}

int main(int argc, char *argv[])
{
  double t;
  if (argc<2){
    printf("Usage: %s <process
number>\n",argv[0]);
    return 1;
  }
  sscanf(argv[1],"%d",&np); mp = 0;
  t = mytime(0);
  if (np<2)
    sum = integrate(f1,a,b,ni);
  else {
    NetInit(np,&mp);
    ShrMemInit();
    myjobp();
    ShrMemDone();
  }
  t = mytime(1);
  printf("time=%lf sum=%22.15le\n",t,sum);
  return 0;
}
```

```
int main(int argc, char *argv[])
{
  double t;
  if (argc<2){
    printf("Usage: %s <thread
number>\n",argv[0]);
    return 1;
  }
  sscanf(argv[1],"%d",&nt); mt = 0;
  t = mytime(0);
  if (nt<2)
    sum = integrate(f1,a,b,ni);
  else {
    ThreadInit();
    ThreadDone();}
  }


  t = mytime(1);
  printf("time=%lf sum=%22.15le\n",t,sum);
  return 0;
}
```

Реализация на OpenMP

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "mycom.h"
#include <omp.h>

int nt, mt;
double a = 0; double b = 1; int ni = 1000000000;
int num = 0; double sum = 0;
double f1(double x);
double f1(double x) { return 4.0/(1.0+x*x); }
void myjobt(int mt);
void myjobt(int mt)
{
  int n1; double a1, b1, h1, s;
  n1 = ni / nt;
  h1 = (b - a) / nt;
  a1 = a + h1 * mt;
  if (mt<nt-1) b1 = a1 + h1; else b1 = b;
  s = integrate(f1,a1,b1,n1);
  printf("mt=%d a1=%le b1=%le n1=%d s1=%le\n",mt,a1,b1,n1,s);
  #pragma omp critical
  {
    sum += s;
    num++;
  } // end critical
  return;
}

int main(int argc, char *argv[])
{
  double t;
  if (argc<2){
    printf("Usage: %s <thread number>\n",argv[0]);
    return 1;
  }
  sscanf(argv[1],"%d",&nt);
```

```
  if (nt<1) nt = 1; mt = 0;
  t = mytime(0);
  if (nt<2)
    sum = integrate(f1,a,b,ni);
  else {
    omp_set_num_threads(nt);
    #pragma omp parallel
    {
      int mt = omp_get_thread_num();
      myjobt(mt);
    } // end parallel
    while (num<nt);
  }
  t = mytime(1);
  printf("time=%lf sum=%22.15le\n",t,sum);
  return 0;
}
```

Трансляция:

```
>mpicc -o ex05a.px -O2 ex05a.c mycom.c -lm
>mpicc -o ex05b.px -O2 -pthread ex05b.c mycom.c -lm
>mpicc -o ex05c.px -O2 -fopenmp ex05c.c mycom.c -lm
```

Результаты выполнения:

```
>ex05a.px 1
time=17.390892 sum= 3.141592651591870e+00

>ex05a.px 2
mp=1 a1=5.000000e-01 b1=1.000000e+00 n1=500000000 s1=1.287002e+00
mp=0 a1=0.000000e+00 b1=5.000000e-01 n1=500000000 s1=1.854590e+00
time=8.698782 sum= 3.141592648390306e+00

>ex05b.px 1
time=17.350918 sum= 3.141592651591870e+00

>ex05b.px 2
mt=1 a1=5.000000e-01 b1=1.000000e+00 n1=500000000 s1=1.287002e+00
mt=0 a1=0.000000e+00 b1=5.000000e-01 n1=500000000 s1=1.854590e+00
time=8.804832 sum= 3.141592648390306e+00

>ex05c.px 1
time=17.350918 sum= 3.141592651591870e+00

>ex05c.px 2
mt=1 a1=5.000000e-01 b1=1.000000e+00 n1=500000000 s1=1.287002e+00
mt=0 a1=0.000000e+00 b1=5.000000e-01 n1=500000000 s1=1.854590e+00
time=8.804832 sum= 3.141592648390306e+00
```