Семинар 2. Создание локальных процессов и простейший обмен данными между ними.

1. Создание локальных процессов.

1) “тяжелых” и “легкие” локальные процессы, обработчик контекста.

Контекст – образ приложения в оперативной памяти компьютера с присвоенным ему системным номером – pid (process identificator), а также все данные, расположенные в стеке и в сегментах динамической памяти.

Процесс – одновременно и контекст приложения, и процесс выполнения кода из этого контекста.

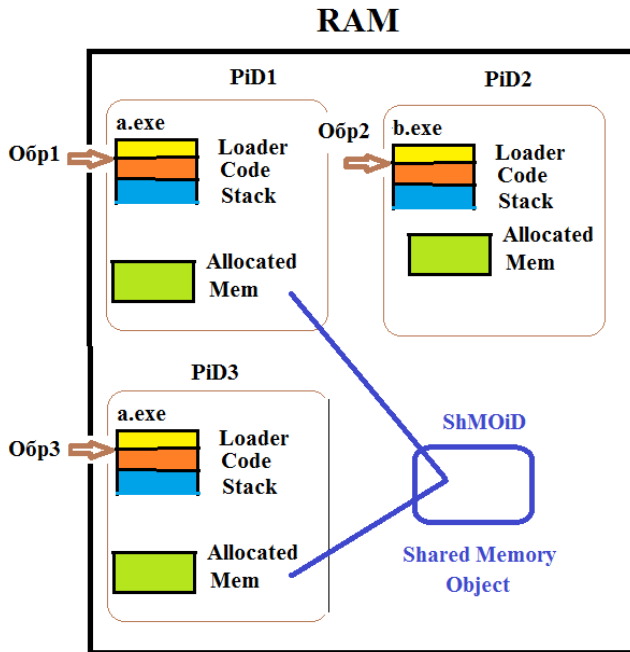
Тяжелый процесс – тоже самое, что процесс.

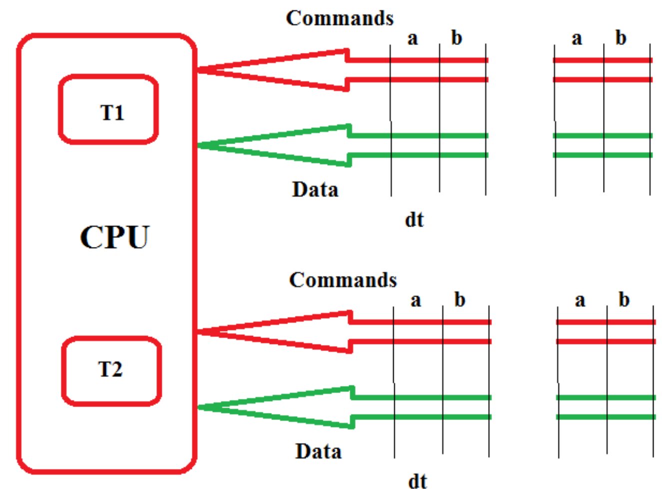
Легкий процесс (трэд, поток) – системный обработчик кода в тяжелом процессе и собственно процесс вычислений, инициируемый этим обработчиком.

Организацией исполнения всех контекстов занимается системный планировщик (sheduler), который формирует очереди команд и данных для всех физических вычислителей (процессоров, ядер, потоков).

Запуск тяжелого процесса:

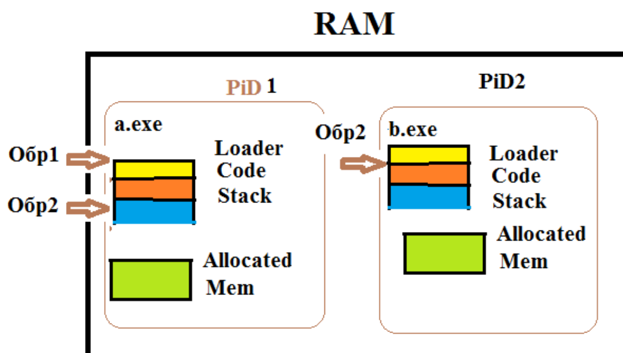
>a.exe
>a.exe
>b.exe

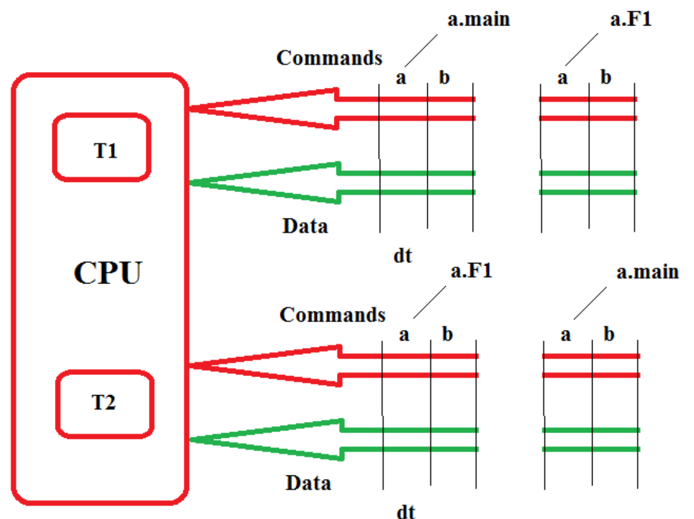


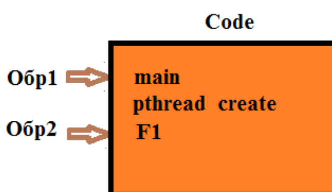


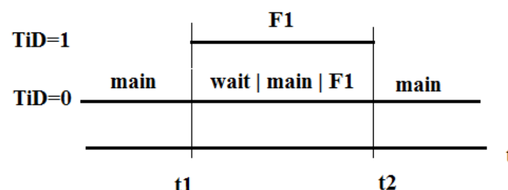
Запуск легкого процесса:

>a.exe
>b.exe

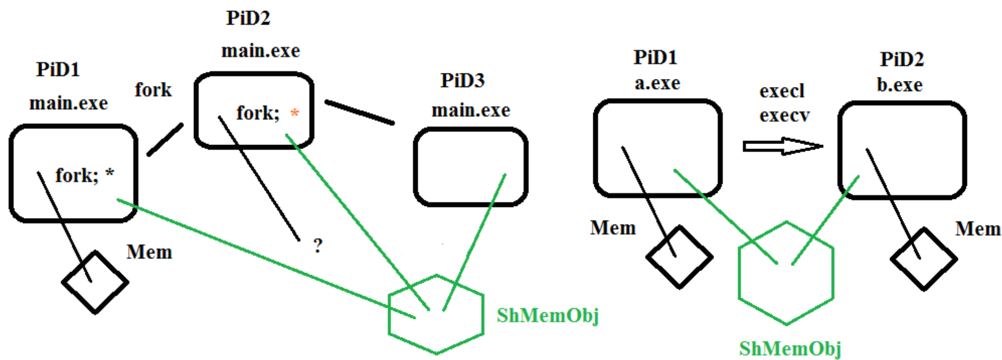






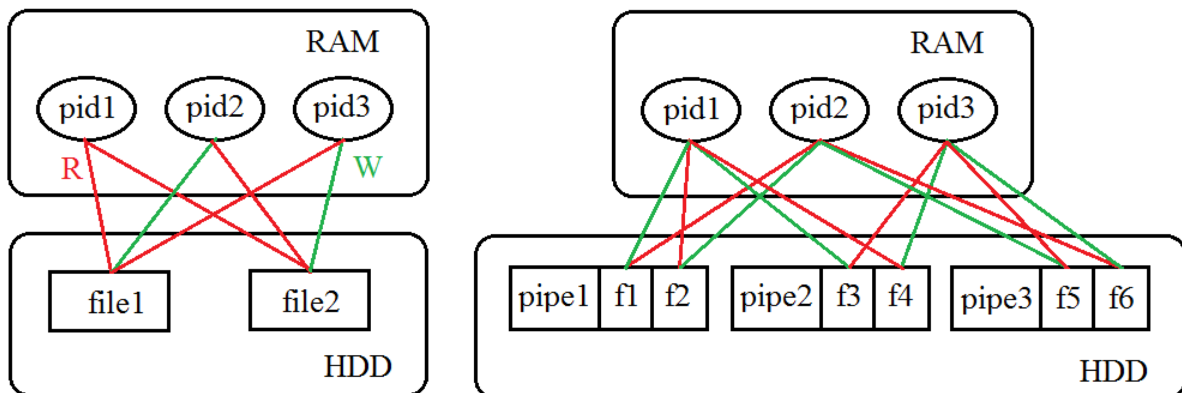


2) Создание “тяжелых” локальных последовательных процессов:
а) Запуск пользователем нескольких программ (main1.px, main2.px, …);
б) Запуск из головной программы нескольких процессов (fork, execl, execv, system, wait).

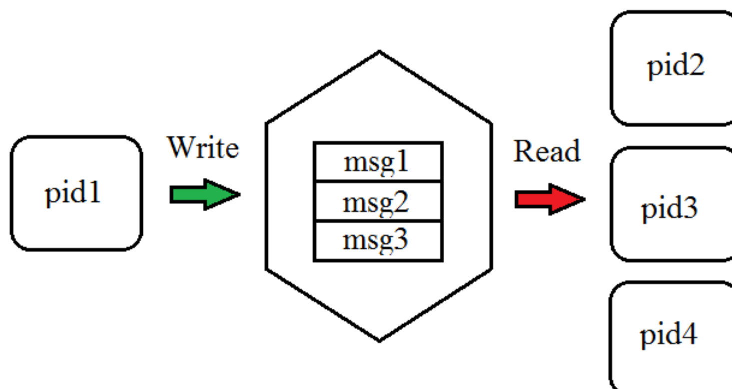


3) Создание “легких” локальных последовательных процессов – ветвление процесса (pthread_create, pthread_join).
4) Гибридный способ – создание процессов обоих типов (server, client, запускающие трэды).

2. Простой обмен данными между локальными процессами.
1) Обмен сигналами (функции kill – посылка сигнала процессу, signal – вызов функции пользователя при получении сигнала с заданным номером).
Функция **kill(pid, sig)** посылает процессу с номером **pid** сигнал **sig**; sig – целочисленное значение (номер) сигнала, первоначально в диапазоне 0..31; sig = 0 – деление на 0, sig = 1 – перезапуск приложения, sig = 9 – ошибка доступа к памяти, sig = 15 – прерывание работы приложения пользователем (Ctrl-C). Для пользовательских целей в системе зарезервированы два номера sig = USER1 и USER2 (первоначально 30 и 31). С их помощью можно создавать собственную систему сигналов (кодирование).
Функция **signal(sig, sig_fun)** позволяет переопределить для заданного sig системный обработчик сигнала на свой (sig_fun).
2) Обмен с помощью разделяемых файлов (функции open, close, read, write, блокировка записи lockf()).
3) Обмен с помощью каналов (функция pipe: два файловых дескриптора для чтения и для записи).



4) Обмен с помощью очередей сообщений (функции msgget, msgsnd, msgrcv, msgctl – порции данных одинаковой длины).



3. Примеры организации локальных параллельных процессов.

Пример 1 – простейшее разветвление процесса (ex03a.c).

Задача – создать параллельное приложение, состоящее из np=5 тяжелых процессов и определить в каждом процессе, кто потомок (slave), а кто родитель (master), и логический номер процесса (mp).

Текст программы:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
void prt_proc(int np, int mp, pid_t pid, pid_t* pids);
void prt_proc(int np, int mp, pid_t pid, pid_t* pids)
{
  int i;
  if (mp==0)
    printf("pid=%d -> I am master, mp=%d\n",pid,mp);
  else
    printf("pid=%d -> I am slave, mp=%d\n",pid,mp);
  for (i=0; i<np; i++)
    printf("pid=%d -> i=%d pids=%d\n",pid,i,pids[i]);
  return;
}
int main(int argc, char *argv[])
{
  pid_t pid, spid=0, pids[10];
  int i, np=5, mp=0;
  pid = getpid();
  printf("pid=%d -> Hello!\n",pid);
  for (i=0; i<np; i++)
    pids[i] = 0;
  i = 0; pids[i] = pid; i++;
  while (i<np){
    if (spid>0 || i==1) spid = fork();
    if (spid>0) pids[i] = spid;
    i++;
    if (spid>0) sleep(1);
  }
  if (spid == 0){
    pid = getpid();
    mp = np;
    while (pids[mp-1] == 0)
      mp--;
  }
  prt_proc(np,mp,pid,pids);
  return 0;
}
```

Трансляция и результат выполнения:

```
Трансляция:
>mpicc -o ex03a.px -O2 ex03a.c -lm
Результат запуска:
>ex03a.px
pid=10450 -> Hello!
pid=10451 -> I am slave, mp=1
pid=10451 -> i=0 pids=10450
pid=10451 -> i=1 pids=0
pid=10451 -> i=2 pids=0
pid=10451 -> i=3 pids=0
pid=10451 -> i=4 pids=0
pid=10452 -> I am slave, mp=2
pid=10452 -> i=0 pids=10450
pid=10452 -> i=1 pids=10451
pid=10452 -> i=2 pids=0
pid=10452 -> i=3 pids=0
pid=10452 -> i=4 pids=0
pid=10453 -> I am slave, mp=3
pid=10453 -> i=0 pids=10450
pid=10453 -> i=1 pids=10451
pid=10453 -> i=2 pids=10452
pid=10453 -> i=3 pids=0
pid=10453 -> i=4 pids=0
pid=10454 -> I am slave, mp=4
pid=10454 -> i=0 pids=10450
pid=10454 -> i=1 pids=10451
pid=10454 -> i=2 pids=10452
pid=10454 -> i=3 pids=10453
pid=10454 -> i=4 pids=0
pid=10450 -> I am master, mp=0
pid=10450 -> i=0 pids=10450
pid=10450 -> i=1 pids=10451
pid=10450 -> i=2 pids=10452
pid=10450 -> i=3 pids=10453
pid=10450 -> i=4 pids=10454
```

Пример 2 – разветвление процесса с обменом данными с помощью очереди сообщений (ex03b.c).

Текст программы:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSG_ID   7777
#define MSG_PERM 00666
#define LBUF     100
typedef struct tag_msg_t{
  int type; pid_t buf[LBUF];
} msg_t;
void prt_proc(int np, int mp, pid_t pid, pid_t* pids);
void prt_proc(int np, int mp, pid_t pid, pid_t* pids)
{
  int i;
  if (mp==0)
    printf("pid=%d -> I am master, mp=%d\n",pid,mp);
  else printf("pid=%d -> I am slave, mp=%d\n",pid,mp);
  for (i=0; i<np; i++)
    printf("pid=%d -> i=%d pids=%d\n",pid,i,pids[i]);
  return;
}
int main(int argc, char *argv[])
{
  pid_t pid, spid=0, pids[LBUF];
  int i, np=5, mp=0;
  int msgid; msg_t msg;
  pid = getpid(); printf("pid=%d -> Hello!\n",pid);
  for (i=0; i<np; i++) pids[i] = 0;

  i = 0; pids[i] = pid; i++;
  while (i<np){
    if (spid>0 || i==1) spid = fork();
    if (spid>0) pids[i] = spid;
    i++;
  }

  if (spid == 0){
    pid = getpid(); mp = np;
    while (pids[mp-1] == 0)
      mp--;
  }

  if (mp == 0){
    msgid = msgget(MSG_ID, MSG_PERM | IPC_CREAT);
    msg.type = 888;
    for(i=0;i<np;i++) msg.buf[i] = pids[i];
    for(i=1;i<np;i++)
     msgsnd(msgid, &msg, sizeof(msg_t), 0);
    sleep(np+1);
    msgctl(msgid, IPC_RMID, (struct msqid_ds *) 0);
    prt_proc(np,mp,pid,pids);
  }
  else{
    while((msgid = msgget(MSG_ID, MSG_PERM)) < 0);
    msgrcv(msgid, &msg, sizeof(msg_t), 0, 0);
    for(i=0;i<np;i++) pids[i] = msg.buf[i];
    sleep(mp);
    prt_proc(np,mp,pid,pids);
  }
  return 0;
}
```

Трансляция и результат выполнения:

```
Трансляция:
>mpicc -o ex03b.px -O2 ex03b.c -lm
Результат запуска:
>ex03b.px
pid=10740 -> Hello!
pid=10744 -> I am slave, mp=1
pid=10744 -> i=0 pids=10740
pid=10744 -> i=1 pids=10741
pid=10744 -> i=2 pids=10742
pid=10744 -> i=3 pids=10743
pid=10744 -> i=4 pids=10744
pid=10742 -> I am slave, mp=2
pid=10742 -> i=0 pids=10740
pid=10742 -> i=1 pids=10741
pid=10742 -> i=2 pids=10742
pid=10742 -> i=3 pids=10743
pid=10742 -> i=4 pids=10744
pid=10741 -> I am slave, mp=3
pid=10741 -> i=0 pids=10740
pid=10741 -> i=1 pids=10741
pid=10741 -> i=2 pids=10742
pid=10741 -> i=3 pids=10743
pid=10741 -> i=4 pids=10744
pid=10743 -> I am slave, mp=4
pid=10743 -> i=0 pids=10740
pid=10743 -> i=1 pids=10741
pid=10743 -> i=2 pids=10742
pid=10743 -> i=3 pids=10743
pid=10743 -> i=4 pids=10744
pid=10740 -> I am master, mp=0
pid=10740 -> i=0 pids=10740
pid=10740 -> i=1 pids=10741
pid=10740 -> i=2 pids=10742
pid=10740 -> i=3 pids=10743
pid=10740 -> i=4 pids=10744
```

Пример 3 – создание легких и тяжелых процессов (ex04a.c, ex04b.c).
Задача численного интегрирования по формуле средних прямоугольников.

$$I=\int_a^b f(x)dx \approx \sum_{i=1}^{N} f\left(x_i\right)h,\quad h=\frac{b-a}{N},\quad x_i=a+h\left(i-0.5\right),\quad i=1,\dots,N;$$

$$I=\sum_{k=0}^{p-1} I_k,\quad I_k=\int_{a_k}^{b_k} f(x)dx,\quad a_k=a+\frac{b-a}{p}k,\quad b_k=\min\left(a_{k+1},b\right),\quad k=0,\dots,p-1;$$

$$I_k \approx \sum_{i=1}^{M} f\left(x_i^k\right)h_k,\quad h_k=\frac{b_k-a_k}{M},\quad x_i^k=a_k+h_k\left(i-0.5\right),\quad i=1,\dots,M,\quad M=\frac{N}{p}.$$

Тяжелые процессы:

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

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSG_ID   7777
#define MSG_PERM 00600
#define LBUF     10
typedef struct tag_msg_t
{ int n; double buf[LBUF]; } msg_t;
static int msgid;
static msg_t msg;

static int np, mp;
static double a = 0;
static double b = 1;
static int ni = 1000000000;
static double sum = 0;

void NetInit(int np, int* mp);
double f1(double x);
double f1(double x){ return 4.0/(1.0+x*x);}

double myjobp(int mp);double myjobp(int mp)
{
  int n1;
  double a1, b1, h1, s1;
  h1 = (b - a) / np;
  n1 = ni / 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);
  return s1;
}

int main(int argc, char *argv[])
{
  int i; 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);
```

Легкие процессы:

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

#include <pthread.h>
static pthread_t* threads;
#define LBUF     10
static double sums[LBUF];

static int nt, mt;
static double a = 0;
static double b = 1;
static int ni = 1000000000;
static double sum = 0;

double f1(double x);
double f1(double x){ return 4.0/(1.0+x*x);}

void* myjobt(void* m); void* myjobt(void* m)
{
  int n1, mt=(int)(((long long int)m)%1024);
  double a1, b1, h1;
  n1 = ni / nt;
  h1 = (b - a) / nt;
  a1 = a + h1 * mt;
  if (mt<nt-1) b1 = a1 + h1; else b1 = b;
  sums[mt] = integrate(f1,a1,b1,n1);
  printf("mt=%d a1=%le b1=%le n1=%d
s1=%le\n",mt,a1,b1,n1,sums[mt]);
  return 0;
}

int main(int argc, char *argv[])
{
  int i; 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);
```

```c
  else {
    NetInit(np,&mp);
    if (mp == 0)
      msgid = msgget(MSG_ID, MSG_PERM |
IPC_CREAT);
    else
      while((msgid = msgget(MSG_ID,
MSG_PERM))<0);
    sum = myjobp(mp);
    if (mp>0){
      msg.n = 1; msg.buf[0] = sum;
      msgsnd(msgid,&msg,sizeof(msg_t),0);
    }
    else{
      for(i=1;i<np;i++){

msgrcv(msgid,&msg,sizeof(msg_t),0,0);
        sum = sum + msg.buf[0];
      }
      msgctl(msgid,IPC_RMID,(struct
msqid_ds *)0);
    }
  }

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

void NetInit(int np, int* mp)
{
  int i; pid_t spid = 0;
  if (np>1){
    i = 1;
    while (i<np){
      if (spid>0 || i==1)
        { *mp=i; spid = fork();}
      if (spid==0) return;
      i++;
    }
  }
  *mp = 0;
  return;
}
```

```c
  else {
    if (!(threads = (pthread_t*)
malloc(nt*sizeof(pthread_t))))
      sys_err("server: not enough memory");
    for (i=0; i<nt; i++)
      if
(pthread_create(threads+i,0,myjobt,(void*)i))
        sys_err("server: cannot create
thread");
    for (i=0; i<nt; i++)
      if (pthread_join(threads[i],0))
        sys_err("server: cannot wait
thread");
      else
        sum = sum + sums[i];
    free(threads);
  }

  t = mytime(1);

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

Трансляция и результаты выполнения:

```
>mpicc -o ex04a.px -O2 ex04a.c mycom.c -lm
>mpicc -o ex04b.px -O2 -pthread ex04b.c mycom.c -lm

>ex04a.px 1
time=18.469721 sum= 3.141592651589794e+00

>ex04a.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=8.688225 sum= 3.141592648389793e+00

>ex04b.px 1
time=18.295300 sum= 3.141592651589794e+00

>ex04b.px 2
mt=0 a1=0.000000e+00 b1=5.000000e-01 n1=500000000 s1=1.854590e+00
mt=1 a1=5.000000e-01 b1=1.000000e+00 n1=500000000 s1=1.287002e+00
time=8.689911 sum= 3.141592648389793e+00
```