В первом семестре курса Операционные системы выполняется 6 лабораторных работ.

## 1. Первая лабораторная работа состоит из двух частей:

- Получение с помощью дизассемблера sr.exe кода прераывания INT 8h.

Студент составляет отчет, состоящий из полученного кода INT 8h, алгоритма 8-го прерывания и алгоритма подпрограммы, которая вызывается в начале и в конце кода 8-го прерывания, в графическом виде по ГОСТу. Студент защищает работу.

- Изучение функций прерывания от системного таймера по литературе. В результате самостоятельной работы студент составляет отчет, в котором перечисляет отдельно функции системного таймера для ОС семейств Windows и Unix/Linux. При этом информация структурируется: по тику, по главному тику и по кванту.

Изучение особенностей пересчета динамических приоритетов для ОС семейств Windows и Unix/Linux. Результаты работы отражаются в отчете. Студент защищает работу. Следует обратить особое внимание на современные ядра Unix/Linux.

## 2. Вторая лабораторная работа – перевод компьютера в защищенный режим:

Лабораторная работа 2
Защищенный режим

Задание:

Написать программу, переводящую компьютер в защищенный режим (32-разраядный режим работы компьютеров на базе процессоров Intel). Программа начинает работать в реальном режиме. Для перевода в защищенный режим выполняются необходимые действия. В защищенном режиме программа работает на нулевом уровне привилегий.

В защищенном режиме программа должна:

- определить объем доступной физической памяти;
- осуществить ввод с клавиатуры строки с выводом введенной строки на экран;
- получить информацию на экране от системного таймера или в виде мигающего курсора, или в виде количества тиков с момента запуска программы на выполнение, или в виде значения реального времени.

Затем программа корректно возвращается в реальный режим с соответствующими сообщениями.

**Для реализации поставленной задачи необходимо:**

- Создать две системные таблицы – глобальную таблицу дескрипторов ( GDT ) для описания сегментов физической памяти, с которыми будет работать запущенная программа и таблицу дескрипторов прерываний ( IDT), в которой заполняются дескрипторы прерываний, которые необходимы для выполнения поставленной задачи.
- Заполнить дескрипторы в обоих таблицах необходимой информацией.
- Заполнить селекторы значениями смещения к соответствующим дескрипторам сегментов.
- Перевести компьютер из реального в защищенный режим, установив флаг ре в 1.
- В защищенном режиме определить объем доступной физической памяти следующим образом – первый мегабайт пропустить; начиная со второго мегабайта сохранить байт или слово памяти, записать в этот байт или слово сигнатуру, прочитать сигнатуру и сравнить с сигнатурой в программе, если сигнатуры совпали, то это – память. Вывести на экран полученной количество байтов доступной памяти.
- Для ввода строки с клавиатуры необходимо написать обработчик прерывания от клавиатуры. Доступ к обработчику осуществляется через пердварительно заполненный дескриптор прерывания от клавиатуры в IDT.
- Для получения информации от системного таймера необходимо написать обработчик прерывания от системного таймера. Доступ к обработчику осуществляется через

предварительно заполненный необходимой информацией дескриптор прерывания от таймера в IDT.

- Если в таблице дескрипторов прерываний были пропущены первые 32 дескриптора (так сделано фирмой Microsoft), то необходимо перепрограммировать контроллер прерывания на новый базовый вектор.
- При переходе в защищенный режим необходимо открыть линию A20.
- Возвращение в реальной режим должно выполняться корректно с использованием 32-разрядных операндов.
- При переходе из режима в режим выдавать соответствующие сообщения.
- В защищенном режиме информация для вывода на экран записывается непосредственно в видеобуфер.
- Для возвращения в реальный режим выполнить необходимые действия.

Литература

1. Рудаков П.И., Финогенов К.Г. Программирование на **ASSEMBLER. – M.:** Диалог МИФИ, 2001, с.640
2. Зубков С. В. Программирование Assembler. - М.: Мир, 2004, с.685

### 3. Третья лабораторная работа. Командная строка ОС UNIX/LINUX

Коротко перечень команд:

Команда ps с ключами ls

# ps –al

Студент рассказывает о информации, получаемой о процессе. Отдельное внимание обращаю на первое поле FLAGS, которое показывает как был запущен процесс:

- 0 – fork() и exec()
- 1 – fork()
- super user
- спрашиваю об особенностях создания процесса и о флаге copy-on-write? super user?

Команда ls –al

Линки: hurd и soft

### 4. Четвертая лабораторная работа. Пять системных вызовов ОС UNIX/LINUX fork(), wait(), exec(), pipe(), signal()

Пример последней пятой программы с собственным обработчиком сигнала. Написана студенткой Титовой.

```
/*
      В программу с программным каналом включить собственный обработчик сигнала.
      Использовать сигнал для изменения хода выполнения программы.
*/

#define _POSIX_SOURCE

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <signal.h>
#include <string.h>

#define MSGSIZE 256


char buf[MSGSIZE];

int parentpipe[2];
```

```
int childpipe[2];

void parentHandler(int sig_numb) {

      int check = 0;

      close(parentpipe[1]);
      char buf_char;
    while (read(parentpipe[0], &buf_char, 1) > 0)
      write(STDOUT_FILENO, &buf_char, 1);
    close(parentpipe[0]);
      return;
}

void childHandler(int sig_numb) {

      close(childpipe[1]);
      char buf_char;
    while (read(childpipe[0], &buf_char, 1) > 0)
      write(STDOUT_FILENO, &buf_char, 1);
    close(childpipe[0]);
      return;
}
int main (void)
 {
      int status;
      // назначение обработчика сигнала
    signal(SIGUSR1, parentHandler);
      signal(SIGUSR2, childHandler);

      // Create the pipe.
      if (pipe (parentpipe)) {
            printf ("Pipe failed.\n");
            return EXIT_FAILURE;
      }


      // Create the pipe.
      if (pipe (childpipe)) {
            printf ("Pipe failed.\n");
            return EXIT_FAILURE;
      }

    // Create the child process.
      pid_t childpid = fork();

      if (childpid == -1) {
            printf("Error: can't fork.\n");
            return 1;
      }
      else if (childpid == 0) {

            close(childpipe[0]);

            write(childpipe[1], "Child says: Yo, I'm OK.\n", strlen("Child says:
Yo, I'm OK.\n"));
            write(childpipe[1], "Child says: Buy.\n", strlen("Child says: Buy.\
n"));
            close(childpipe[1]);

            kill(getppid(), SIGUSR2);

            return EXIT_SUCCESS;
```

```
      }
      else {
            close(parentpipe[0]);
            write(parentpipe[1], "Parent says: Hello, how are you?\n",
strlen("Parent says: Hello, how are you?\n"));
            write(parentpipe[1], "Parent says: Haven't seen you for ages.\n",
strlen("Parent says: Haven't seen you for ages.?\n"));

            close(parentpipe[1]);
            kill(childpid, SIGUSR1);

            wait(&status);
            return EXIT_SUCCESS;
     }
}
```

5. **Взаимоисключение при взаимодействии параллельных процессов. Семафоры и разделяемая память**
   - **Производство - потребление**

```
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

const int MY_FLAG = IPC_CREAT | S_IRWXU | S_IRWXG | S_IRWXO;
const int num = 10;
const int count = 15;

int* buf;
int* pos;

#define POTR  (0)
#define PROIS (1)
#define BIN   (2)

struct sembuf start_pr[2] = {{PROIS,-1,0},{BIN,-1,0}};
struct sembuf stop_pr[2]  = {{POTR,1,0},{BIN,1,0}};
struct sembuf start_po[2] = {{POTR,-1,0},{BIN,-1,0}};
struct sembuf stop_po[2]  = {{PROIS,1,0},{BIN,1,0}};
??????????
int main()
{

      key_t mem_key;
      my_key = ftok("my_file",0);
      int shmid, semid, status;
      if ((shmid = shmget(mem_key, (num + 1)*sizeof(int), MY_FLAG)) == -1)
      {
            printf("Error in shmget\n");
            return 1;
      }
      pos = shmat(shmid, 0, 0);
      buf = pos + sizeof(int);

      (*pos) = 0;
      if (*buf == -1)
      {
            printf("Error in shmat\n");
```

```
            return 1;
      }
      if ((semid = semget(my_key, 3, MY_FLAG)) == -1)
      {
            printf("Error in semget\n");
            return 1;
      }
      semctl(semid, 2, SETVAL, 0);
      semctl(semid, 1, SETVAL, num);
      semctl(semid, 2, SETVAL, 1);

      srand(time(NULL));

      pid_t pid;
      if ((pid = fork()) == -1)
      {
            printf("Error in fork\n");
            return 1;
      }
      int k = 0;
      switch(pid == 0)
      {
        case 0:
          while(k < count)
            {
                  semop(semid, start_pr, 2);
                  buf[*pos] = 1+rand()%10;
                  printf("\tProisvodstvo^ [%d] -- [%d]\n", k, buf[*pos]);
                  (*pos)++;
                  semop(semid, stop_pr, 2);
                  sleep(rand()%2);
                  k++;
            }
      break;
        default:
            while (k < count)
            {
                  semop(semid, start_po, 2);
                  (*pos)--;
                  printf("Potreblenie^  [%d] -- [%d]\n", k, buf[*pos]);
                  semop(semid, stop_po, 2);
                  sleep(rand()%2);
                  k++;
            }
      break;
      }
      if (pid != 0)
      {
            if (shmdt(pos) == -1)
            {
                  printf("Error in shmdt\n");
                  return 1;
            }
         wait(&status);
      }
      return 0;
}
```

- **Монитор Хоара «Читатели-писатели»**

## 6. Реализация монитора Хоара «Читатели-писатели» под ОС Windows

Разрабатывается многопоточное приложение под ОС Windows с использованием объектов ядра: event и mutex. Потоки разделяют глобальную переменную.