Операционные системы

лекции 9, 10

- Понятие процесса.
- Порождение процесса.
- Механизм замены тела процесса (exec)
- Схема запуска одной программы из другой (fork-exec-wait). Анализ успешности завершения процесса.
- Формирование 0 и 1 процессов.

3.11.2010

Определения (повторение)

Вычислительная система (ВС) - совокупность аппаратных и программных средств, функционирующих как единое целое и предназначенная для решения определенного класса задач.

Операционная система (ОС) - комплекс программ осуществляющий управление, распределение и контроль за использованием ресурсов вычислительной системы

Процесс - совокупность данных и машинных команд, исполняющаяся в рамках ВС и обладающую правами на владение некоторым набором ресурсов.

Управление процессами

Управление процессами включает следующие задачи:

- обеспечение жизненного цикла процессов (создание, выполнение и уничтожение)

- распределение ресурсов

- синхронизация

- организация взаимодействия процессов.

Понятие контекста процесса.

Типы процессов

Полновесные

Такие процессы имеют собственное адресное пространство для статических и динамических данных

ОС следит за защитой памяти одного полновесного процесса от другого.

Легковесные (нити, threads)

Имеют одну выделенную область памяти (общее виртуальное адресное пространство). Но у каждой нити свой счетчик команд, стек, регистры, нити-потомки, состояние.

Жизненный цикл процесса

ПОРОЖДЕНИЕ создан, но не готов к запуску,
создаются информационные структуры, описывающие процесс
загружается кодовый сегмент процесса

ГОТОВНОСТЬ

ВЫПОЛНЕНИЕ — **цикл**

ОЖИДАНИЕ

ЗАВЕРШЕНИЕ – процесс выгружается из памяти и разрушаются структуры данных, связанные с ним.

Параллельные процессы

Процессы, выполнение которых хотя бы частично перекрывается по времени, называются параллельными.

Важнейшее требование мультипрограммирования:
результат выполнения процессов не должен зависеть от порядка переключения выполнения между процессами, то есть от скоростей выполнения процессов.

Ситуация гонок (race conditions)

Реализация процессов в UNIX

Процесс регистрируется в таблице процессов ядра, получает свой уникальный номер (PID) — идентификатор процесса.

PID==0 процесс ядра **PID==1 процесс init**

Процесс имеет

1. **Сегмент кода**
2. **Сегмент данных, включая стек**
3. **Состояние регистров процессора (PC, PSW)**
4. **ТОФ, буферы ввода/вывода**
5. **Командная строка (аргументы передаются процессу в спец. структуре)**
6. **Переменные окружения**
7. **Текущий каталог, корневой каталог**
8. **Информация об обработке сигналов**
9. **umask**
10. **счетчики потребленных ресурсов**
11. **информация о владельце**

Создание новых процессов

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

ПРИМЕР:

```
int main(){
 int a,pid;
  pid=fork();
  if (pid==-1) {perror("fork"); return 1;}
  if (pid==0){ /*потомок*/
   printf("child\n");
  }
  else{
   printf("parent\n");
  }
  printf("all\n");
  return 0;
}
```

Получение идентификатора процесса

```
pid_t getpid(void);
pid_t getppid(void);
-----------------------------------
int main(){
 int a,pid;
  pid=fork();
  if (pid==-1) {perror("fork"); return 1;}
  if (pid==0){ /*потомок*/
   его номер getpid();
   номер родителя getppid();
  }
  else{/*родитель*/
   его номер  getpid();
   номер потомка - результат fork (в перем. pid)
  }
  return 0;
}
```

Завершение процесса

Варианты завершения процесса

```
#include <unistd.h>
void _exit(int exitcode);
```

О порядке завершения родителя и потомка
Процессы-зомби

Получение статуса завершения потомка родителем

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
```

```
WIFEXITED(*status)
WEXITSTATUS(*status)
```

Механизм замены тела процесса

```
int execve(const char* path,char * const argv[],
                            char * const env[]);
```

```
int execv(const char* path,char * const argv[]);

int execvp(const char* path,char * const argv[]);
```

Примеры использования:

```
char* a[5] = {"gcc","-ansi","-Wall","test.c",NULL}

execv("/bin/gcc",a);

execvp("gcc",a);
```

Механизм замены тела процесса (прод.)

```
int execl(const char* path,const char * arg0,...);

int execlp(const char* path,const char* arg0,...);
```

Примеры использования:

```
execl("/bin/gcc","gcc","-Wall","f.c", NULL);

execlp("gcc","gcc","-Wall","f.c", NULL);
```

Общая схема использования fork-exec

```
int main(){
  int p;
  p=fork();
  if (p==0){
    exec...
    perror("exec");
    return 1;
  }
  else{
    /*процесс-родитель*/
    ....
    wait...
    ....
  return 0;
  }
}
```

Пример: компиляция и запуск программы

(найти ошибки!)

```
int main(int argc, char**argv){
  int p,status;
  if (argc<2) return 1;
  p=fork();
  if (p==0){
    execlp("gcc","gcc",argv[1],"-o","res");
    perror("exec");
    return 1;
  }
  else{/*процесс-родитель*/
    wait(&status);
if ((WIFEXITED(status)&&WEXITSTATUS(status))
    execl("./res","res");
 return 1;
   }
}
```

Управление свойствами процесса

Смена текущего и корневого каталогов

```
int chdir(const char* path);
int chroot(const char* path);
```

Работа с переменными окружения

```
extern char * * environ;

char* getenv(const char* name);
int setenv(const char* name, const char*  value, int wr);
void unsetenv(const char* name);
```

Параметр umask

```
int umask(int mask);
```

Начальная загрузка ОС UNIX. Формирование процессов 0 и 1

Этапы

1.Аппаратно читается 0 блок системного устройства

2.Поиск и считывание в память файла `/unix`, расположенный в корневом каталоге и который содержит код ядра системы

3.Запускается на исполнение этот файл

Процесс 0 не имеет сегмента кода, предст. собой структуру данных. Существует в течение всего времени работы системы.

Процесс 1 сначала создается как копия процесса 0, затем увелич. его область памяти, в его сегмент кода копируется exec. Он необходим для выполнения /etc/init, который запускается и настраивает остальные системные процессы.