«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»

Экзаменационный лист

«12» 01 2021 г.	дисциплина ОС
начало 9 : 00	билет № _____ группа ИУ7-52Б
окончание _____ : _____	студент Тарасов Никита Алексеевич
оценка _____	экзаменатор Рязанова Наталья Юрьевна (подпись)

Вариант 4.

~~Повторно~~ -

1. Тупики: Обнаружение тупиков для повторно используемых ресурсов методов редукции графа, способы представления графа, алгоритмы обнаружения тупиков. Пример анализа состояния системы методом редукции графа. Методы восстановления работоспособности системы.

Повторно-используемые ресурсы - кол-во в системе постоянно и при использовании они не изменяются: аппаратура, реентерабельные коды, системные таблицы, процедуры ОС.

Потребляемые ресурсы - количество в ОС переменно и произвольно: сообщения. Процесс может создать любое кол-во сообщений.

Тупик - ситуация, которая возникает в результате монопольного использования разделяемых ресурсов, когда процесс, владея ресурсом, запрашивает другой ресурс, занятый непосредственно или через цепочку запросов другим процессом, ожидающим освобождения ресурса, занятого первым процессом.

В теории тупиков принято различать два типа ресурсов:

- повторно-используемые ресурсы
- потребляемые ресурсы

Экзаменационный лист

« ____ » _____ 20 ____ г. | дисциплина _____

начало _____ : _____ | билет № _____ группа _____

окончание _____ : _____ | студент _____

оценка _____ | экзаменатор _____ подпись

Условия возникновения тупика:

• Взаимоисключение. Возникает, когда процесс монопольно использует ресурсы

• Ожидание. Когда процессы удерживают занятые ими ресурсы, ожидая предоставления им дополнительных ресурсов.

• Неперераспределяемость. Когда ресурсы нельзя отобрать у процесса до их завершения.

• Круговое ожидание. Возникает замкнутая цепь запросов процессов на дополнительные ресурсы.

Методы борьбы с тупиками

1. Предотвращение или недопущение
2. Обход или недопущение
3. Обнаружение и восстановление

Обнаружение происходит с помощью графовой модели. Система может быть описана двудольным направленным графом. Множества вершин: процессы и ресурсы. Дуга из ресурсов к процессам — выделение или приобретение ресурсов. Дуга из вершины процессов к ресурсам называется запросом.

« ____ » _____ 20 ____ г. | дисциплина _____

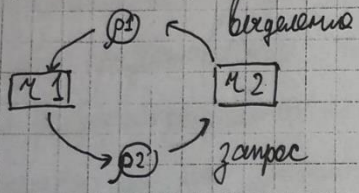
начало _____ : _____ | билет № _____ группа _____

окончание _____ : _____ | студент _____

оценка _____ | экзаменатор _____ подпись

Графовая модель Холта

- приобретение - (ч, р)
- запрос - (р, ч)



Обнаружение тупиков по графовой модели выполняется методом редукции графа. Этот метод основан на предположении, что незаблокированный в тупике процесс может запрашивать приобретения любых нужных ресурсов, а затем освобождать их. Освободившиеся ресурсы могут быть выделены или распределены другими процессами, которые их ожидают.

~~Способы~~ Методы прямого обнаружения.

Последовательно рассматриваются запросы каждого процесса и определяется, может ли запрос быть удовлетворен.

Способы представления графа.

Двудольный граф может быть описан с матрицами или с связными списками. Матричное представление более удобно.

« ____ » _____ 20 ____ г.
начало _____ : _____
окончание _____ : _____
оценка _____

дисциплина _____
билет № _____ группа _____
студент _____
экзаменатор _____ подпись

Восстановление системы.

1. Перезагрузка системы
   - Дорогое решение, так как будет потеряна вся проделанная незавершенная процессами работа
   - Для системы, работающей долгое время, невозможно
2. Последов. завершить процессы попавшие в тупик.
   - В результате, процессы, кот. (работа) завершатся, вернут занимаемые ресурсы
   - Проделанная работа - потеряна
3. Завершать процесс, не имеющие отношения к тупику.
   - решается на основе приоритетов

~~- Отличия~~

Вопрос 2. Задача „Обедающие философы" - модели распределения ресурсов ВС. Известенные семафоры UNIX: сис. вызовы, поддержка в системе, пример из л/р. „производство - потребление".

На круглом столе лежит 5 тарелок, напротив каждой тарелки-философ. Какое-то время философ думает, какое-то ест. Есть философ может только с помощью двух приборов. У тарелки изначально 1 прибор. Три способа действий:

1. Каждый пытается взять две вилки и есть, если может

ч

начало _____ : _____ | билет № _____ группа _____

окончание _____ : _____ | студент _____

оценка _____ | экзаменатор _____ подпись

2. Философ пытается взять правую вилку, после чего, удерживая ее пытаться взять левую

3. -//- (левая и правая поменять местами)

1 способ - бесконечное откладывание. Какой-то философ может умереть голодной смертью.

2 способ - все философы одновременно берут правую вилки и блокируют друг друга

3 способ - -//-

Современные ОС поддерживают наборы считающих семафоров. Самое важное св-во набора семафоров: одной неделимой командой можно изменить все или часть семафоров набора.

Набор семафора:

- Имя
- UID
- Права доступа
- Кол-во семафоров
- Время изменения семафора последним процессом
- Время измен. управ. параметров
- указатель на массив семафоров

начало _____ : _____ | билет № _____ группа _____

окончание _____ : _____ | студент _____

оценка _____ | экзаменатор _____ подпись

О каждом семафоре известно:

- значение
- ID процесса, кот. оперировал с ним последний
- число процессов заблокированных на семафоре

На семафоре определено 3 операции:

- захват, декремент (sem_op<0)
- освобождение, инкремент (sem_op>0)
- проверка семафора на 0. (sem_op=0)

Структура:

```
struct sembuf
{
    n_short  sem_num;   // index
    short    sem_op;    // операция
    short    sem_flg;   // флаги
}
```

sem_op >0 IPC_NOWAIT

sem_op <0 SEM_UNDO

sem_op =0

Взаимоисключение, организация монопольного доступа процесса к разделяемой переменной.

Синхронизация, когда процесс заинтересован в действиях другого процесса.

«___»

начало ___:___ | билет № ___ группа ___

окончание ___:___ | студент ___

оценка ___ | экзаменатор ___ подпись

Задача производство-потребление, если потребитель работает быстрее производителя возникает ситуация, когда буфер пуст, потребитель будет ожидать когда производитель положит что-нибудь в буфер, это видно при передаче сообщений, сообщения несут информацию, кот интересует процесс, для того, чтобы продолжить свое выполнение.

```
#define count_producer 3
#define count_consumer 3
#define n 20

#define sem_bin 0
#define sem_empty 1
#define sem_full 2

#define P -1
#define V 1

struct sembuf producer_start[2] = {
        {sem_empty, P, sem_undo},
        {sem_bin, P, sem_undo}
};

struct sembuf producer_stop[2] = {
        {sem_bin, V, sem_undo},
        {sem_full, V, sem_undo}
};

struct sembuf consumer_start[2] = {
        {sem_full, P, sem_undo}
        {sem_bin, P, sem_undo}
};
```

« ____ » _____ 20 ____ г. | дисциплина _____

начало _____ : _____ | билет № _____ группа _____

окончание _____ : _____ | студент _____

оценка _____ | экзаменатор _____ подпись

```
struct sembuf consumer_stop[2] = {
    {sem_bin, V, sem_undo},
    {sem_empty, V, sem_undo}
};
int sem_id = -1;
int shm_id = -1
int *shm = NULL; int *shm_prod = NULL; int * shm_cons = NULL;

void producer(const int id)
{
    while(1)
    {
        sleep(1);
        perror();
        exit(1);
    }  if (semop(sem_id, produser_start, 2) == -1)
            exit(1);
        *shm + *shm_prod) = *shm_prod;
        (*shm_prod)++;
        if( semop(sem_id, producer_stop, 2) == -1)
            exit(1);
    }
}
void consumer(const int id)
{
    while(1)
    {
        if( semop(sem_id, consumer_start, 2) == -1)
            exit(1);
        (*shm_cons)++;
        if (semop(sem_id, consumer_stop, 2) == -1)
            exit(1);
    }
}
```

нчание _____ : _____ студент _____

нка _____ экзаменатор _____ _____
подпись

```
void catch_sig(int sig_num)
{
    signal(sig_num, catch_sig)
    shmctl(shm_id, IPC_RMID, NULL)
    semctl(sem_id, 0, IPC_RMID, 0)
}
int main(void)
{
    sem_id = semget(IPC_PRIVATE, 3, IPC_CREAT | perms);
    if (sem_id == -1)
        exit(1);

    if (semctl(sem_id, sem_bin, SETVAL, 1) == -1 ||
        semctl(sem_id, sem_empty, SETVAL, N) == -1 ||
        semctl(sem_id, sem_full, SETVAL, 0) == -1)
            exit(1)

    shm_id = shmget(IPC_PRIVATE, (N+3) * sizeof(int), IPC_CREAT |
                                                        perms)
    if (shm_id == -1)
        exit(1)
    shm_prod = shm;
    shm_cons = shm+1;
    *shm_prod = 0;
    *shm_cons = 0
    shm = shm + 2;
    for (int i=0; i < count_producer; ++i)
    {
        const pid_t pid = fork();
        if (pid == -1)
            exit(1);
        else if (pid == 0)
        {
            producer(i);
            exit(1);
        }
    }
```

« ____ » _____ 20 ____ г. | дисциплина _____

начало _____ : _____ | билет № _____ группа _____

окончание _____ : _____ | студент _____

оценка _____ | экзаменатор _____ _____ подпись

```
for(int i=0; i<count_consumer; ++i)
{
    -//-   (producer -> consumer заменить)
}
signal(sigint, catch_sig);
for(int i=0; i< cnt_cons + cnt_prod; ++i)
{   int status;
    const pid_t child_pid = wait(&status)
    if (wifexited(status))
        print(' process %d returns %d', child-pid, wexitstatus(
    else if (wif signaled (status))                    status)
            -//-
    else if wif stopped(status))
            -//-
}
shmctl(shm_id, IPC_RMID, NULL);
semctl(sem_id, sem_bin, IPC_RMID, 0);
return 0;
}
```