Лабораторная работа №1 по курсу: Операционные системы

Выполнил студент группы М8О-206Б-17 МАИ *Новиков Павел Сергеевич.*

Цель работы

Получение навыков использования утилиты strace.

Задание

Часть 1. Написать собственную программу, которая демонстрирует работу с различными вызовами (8-15) операционной системы. Произвести диагностику работы написанной программы с помощью утилит ОС, изучив основные принципы применения используемых утилит.

Часть 2. Выбрать стороннее программное обеспечение. Произвести диагностику ПО. Выявить ключевые особенности работы. Выявить предполагаемые ключевые системные вызовы, которые используются в стороннем программном обеспечении.

Информация

Системные вызовы – один из уровней абстракции, предоставляемый пользовательским программам. Само управление ресурсами проходит незаметно для пользователя и осуществляется в автоматическом режиме. Любой однопроцессорный компьютер одномоментно может выполнить только одну команду. Когда процесс выполняет пользовательскую программу в режиме пользователя и нуждается в некоторой услуге ОС, он должен выполнить команду системного прерывания, чтобы передать управление ОС. Затем ОС по параметрам вызова определяет, что именно требуется вызывающему процессу. После этого она обрабатывает системный вызов и возвращает управление той команде, которая следует за системным вызовом. В некотором смысле выполнение системного вызова похоже на выполнение особой разновидности вызова процедуры, с той лишь разницей, что системные вызовы входят в ядро, а процедурные – нет. Использованные системные вызовы:

Использованные системные вызовы

1. **int** creat(**const char** *pathname, modet mode); – создает и открывает файловый дескриптор в соответствии с флагами открытия.

2. **int** open(**const char** *path, **int** oflag, ...); – открывает файловый дескриптор: первый аргумент – путь до файла, второй – флаги открытия. */

3. **int** read(**int** fd, **void** *buffer, **int** nbyte); – читает nbyte из файлового дескриптора fd в буффер buffer.

4. **int** write(**int** fd, **void** *buffer, **int** nbyte); – записывает количество байтов в 3 аргументе из буфера в файл с дескриптором fd, возвращает количество записанных байтов или -1 в случае ошибки.

5. **int** close(**int** fd); – закрывает файловый дескриптор.

6. **int** fsync(**int** fd); – синхронизирует состояние файла в памяти с его состоянием на диске

7. **int** truncate(**int** fd, offt lenght); – устанавливает длину файла с файловым дескриптором fd в lenght байт. Если файл до этой операции был длиннее, то отсеченные данные теряются. Если файл был короче, то он увеличивается, а добавленная часть заполняется нулевыми байтами.

8. pidt fork(**void**); – создает дочерний процесс. Если возвращает 0, то созданный процесс – ребенок, если >0, то – родитель.

9. pidt wait(**int** *status); – приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.

10. offt lseek(**int** fd, offt offset, **int** whence); – устанавливает смещение для файлового дескриптора в значение аргумента offset в соответствии с директивой whence, которая может принимать одно из следующих значений: SEEKSET(смещение устанавливается в offset байт от начала файла), SEEKCUR(смещение устанавливается как текущее смещение плюс offset байт), SEEKEND(смещение устанавливается как размер файла плюс offset байт).

11. **int** execl(**const char** *path, **const char** *arg); – передаем аргументы в командную строку (когда их количество известно)

12. **void** exit(**int** status); –выходит из процесса с заданным статусом.

Лабораторная работа 2:

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

int fd[2];

#define READ    0
#define WRITE   1
```

```
int factorial (int n) {
    if (n == 0) return 1;
    pid_t pid;
    int res;
    pid = fork();
    if (pid < 0) {
        perror("fork");
    } else if (pid > 0) {
        wait(0);
        read(fd[READ], &res, sizeof(res));
    } else if (pid == 0) {
        res = factorial(n - 1);
        close(fd[READ]);
        if (write(fd[WRITE], &res, sizeof(res)) < 0) {
            perror("write");
        }
        exit(0);
    }
    return n * res;
}

int main() {
    if (pipe(fd) < 0) {
        perror("pipe");
        return 0;
    }
    int n = 0;

    printf("Enter N > 0 to calculate Factorial(N)\n");
    scanf("%d", &n);
    if (n < 0) {
        printf("Bad input\n");
        return 0;
    }
    printf("Factorial(N) equals:\n%d\n", factorial(n));

    return 0;
}
```

Лабораторная работа 3:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <pthread.h>

void print_matrix(int**, int);
void floyd(int**, int, int);
void *floyd_parallel(void*);
int min(int x, int y) {
    if (x < y) {
        return x;
    }
    return y;
}
typedef struct _args {
    int i, k;
    int** matrix;
    int n;
```

```
} Args;

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Use ./prog \"thread count\"\n");
        return 0;
    }
    int thread_count = atoi(argv[1]);
    if (thread_count < 1) {
        printf("Wrong thread count: %d\n", thread_count);
        return 0;
    }
    printf("Enter number of vertexes:\n");
    int n;
    scanf("%d", &n);
    if (n <= 0) {
        printf("Vertex count must be > 0\n");
        return 0;
    }

    printf("Enter adjency matrix:\n");
    int **adjency_matrix = malloc(n * sizeof(*adjency_matrix));
    for (int i = 0; i < n; ++i) {
        adjency_matrix[i] = malloc(n * sizeof(int));
        for (int j = 0; j < n; ++j) {
            scanf("%d", &adjency_matrix[i][j]);
            if (adjency_matrix[i][j] < 0)
                adjency_matrix[i][j] = INT_MAX / 2;
            if (i == j)
                adjency_matrix[i][j] = 0;
        }

    }

    floyd(adjency_matrix, n, thread_count);

    int u = 0, v = 0;
    do {
        --u;
        --v;
        if ((u >= 0) && (v >= 0) && (v < n) && (u < n)) {
            int path = adjency_matrix[u][v];
            if (path == INT_MAX) {
                printf("Path from v%d to v%d doesnt exist\n", u + 1, v + 1);
            } else {
                printf("Shoret path length from v%d to v%d equals: %d\n", u + 1, v + 1, path);
            }
        }
        printf("Enter start & end vertex to show shortest path length:\n");
    } while (scanf("%d%d", &u, &v) != EOF);
    for (int i = 0; i < n; ++i) {
        free(adjency_matrix[i]);
    }
    free(adjency_matrix);
}

void print_matrix(int** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            printf("%d ", matrix[i][j]);
        printf("\n");
    }
    return;
}
```

```
void floyd(int** matrix, int n, int thread_count) {
    thread_count = min(thread_count - 1, n);
    pthread_t *t_id = malloc(thread_count * sizeof(pthread_t));
    Args *args = malloc(thread_count * sizeof(Args));
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < thread_count; ++i) {
            args[i].i = i;
            args[i].k = k;
            args[i].matrix = matrix;
            args[i].n = n;
            pthread_create(&t_id[i], NULL, floyd_parallel,(void*) &args[i]);
        }

        for (int i = thread_count; i < n; ++i) {

            for (int j = 0; j < n; ++j) {
                matrix[i][j] = min(matrix[i][j], matrix[i][k] + matrix[k][j]);
            }
        }
        for (int i = 0; i < thread_count; ++i) {
            pthread_join(t_id[i], NULL);
        }


    }
    free(t_id);
    free(args);
    return;
}

void *floyd_parallel(void* args) {
    Args *ptr = (Args*)args;
    for (int j = 0; j < (ptr -> n); ++j) {
        (ptr -> matrix)[ptr -> i][j] = min((ptr -> matrix)[ptr -> i][j], (ptr -> matrix)[ptr -> i][ptr -> k
    }
    pthread_exit(0);
}
```

Лабораторная работа 4:

```
#include <stdio.h>
#include <ctype.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>

int *mapped_memory;
#define READ    0
```

```
#define WRITE   1

#define SIZE sizeof(int)


int factorial (int n) {
    if (n == 0) return 1;
    pid_t pid;
    int res;
    pid = fork();
    if (pid < 0) {
        perror("fork");
    } else if (pid > 0) {
        wait(0);
        res = *mapped_memory;
    } else if (pid == 0) {
        res = factorial(n - 1);
        *mapped_memory = res;
        exit(0);
    }
    return n * res;
}

int main() {

    int fd = shm_open("/tmp:memory", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
        if(fd == -1){
                perror("shm::open_fail");
                exit(-1);
        }
        if(ftruncate(fd, SIZE) == -1){
                perror("trucate::fail");
                exit(-1);
        }

        mapped_memory = mmap(NULL, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
        if(mapped_memory == MAP_FAILED){
                perror("mmap::mapping_fail");
                fprintf(stderr, "%p", mapped_memory);
                exit(-1);
        }
        close(fd);

    int n = 0;

    printf("Enter N > 0 to calculate Factorial(N)\n");
    scanf("%d", &n);
    if (n < 0) {
        printf("Bad input\n");
        return 0;
    }
    printf("Factorial(N) equals:\n%d\n", factorial(n));

    return 0;
}
```

Лабораторная работа 5:

```
#include <stdio.h>
#include "list.h"
```

```
int main(){
        List* list = d_create();
        d_push_back(list ,"bwa");
        d_push_back(list ,"rtt");
        d_push_back(list ,"gdb");
        d_pop_front(list);
        d_push_front(list ,"abc");
        d_push_front(list ,"awd");
        d_pop_back(list);
        d_print(list);
        d_destroy(list);
}
```

Диагностика программ с помощью strace

Лабораторная работа 2:

```
Enter N > 0 to calculate Factorial(N)
5
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f4f38f02810)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=16829, si_uid=1000, si_status=0,\ si_utime=0, si_
Factorial(N) equals:
120
+++ exited with 0 +++
```

Лабораторная работа 3:

```
execve("./prog", ["./prog", "10"], 0x7fffe41883e8 /* 49 vars */) = 0
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff960bb7b0) = -1 EINVAL (Invalid argument)
arch_prctl(ARCH_SET_FS, 0x7fa0c7ff3740) = 0
Enter number of vertexes:
3
Enter adjency matrix:
1 1 1
1 1 1
1 1 1
clone(child_stack=0x7fa0c7ff1fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
clone(child_stack=0x7fa0c77f0fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
clone(child_stack=0x7fa0c6fd5fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
clone(child_stack=0x7fa0c6fd5fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
clone(child_stack=0x7fa0c77f0fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
clone(child_stack=0x7fa0c7ff1fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
clone(child_stack=0x7fa0c7ff1fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
clone(child_stack=0x7fa0c77f0fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
clone(child_stack=0x7fa0c6fd5fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSV
Enter start & end vertex to show shortest path length:
1 5
Enter start & end vertex to show shortest path length:
exit_group(0)                           = ?
+++ exited with 0 +++
```

Лабораторная работа 4:

```
mmap(NULL, 218437, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2c9252d000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2c9252b000
mmap(NULL, 39416, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f2c92521000
mmap(0x7f2c92523000, 16384, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f2c92
mmap(0x7f2c92527000, 8192, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7f2c92527000
mmap(0x7f2c92529000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7f2c92
mmap(NULL, 1852992, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f2c9235c000
mmap(0x7f2c9237e000, 1359872, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7f2
mmap(0x7f2c924ca000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16e000) = 0x7f2c924ca000
```

```
mmap(0x7f2c92517000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba000) = 0x7f2
mmap(0x7f2c9251d000, 13888, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f2c9251c
mmap(NULL, 131528, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f2c9233b000
mmap(0x7f2c92341000, 61440, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7f2c92
mmap(0x7f2c92350000, 24576, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x15000) = 0x7f2c92350000
mmap(0x7f2c92356000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a000) = 0x7f2c9
mmap(0x7f2c92358000, 12744, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f2c92358
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2c92338000
mmap(NULL, 4, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f2c9258c000
Enter N > 0 to calculate Factorial(N)
4
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=19219, si_uid=1000, si_status=0, si_utime=0, si_s
Factorial(N) equals:
24
+++ exited with 0 +++
Time: 0h:00m:08s
```

Лабораторная работа 5:

```
dynlink:
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/tls/haswell/x86_64/liblist.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or
openat(AT_FDCWD, "/usr/lib/tls/haswell/liblist.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or direct
openat(AT_FDCWD, "/usr/lib/tls/x86_64/liblist.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directo
openat(AT_FDCWD, "/usr/lib/tls/liblist.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/haswell/x86_64/liblist.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or di
openat(AT_FDCWD, "/usr/lib/haswell/liblist.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64/liblist.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/liblist.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
./dynlink: error while loading shared libraries: liblist.so: cannot open shared object file: No such file o
+++ exited with 127 +++
dynload:
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/home/mika/lab/os/lab05/liblist.so", O_RDONLY|O_CLOEXEC) = 3
```

Выводы

Утилиты диагностики позволяют быстро выяснить, как программа взаимодействует с операционной системой. Это происходит путем мониторинга системных вызовов и сигналов. Становится понятнее работа собственного и стороннего программного обеспечения. Можно узнать, что делают конкретные системные вызовы, подсчитать их, узнать время их работы и ошибки при их использовании, если они есть, отследить запущенные процессы. В случае, если у нас нет доступа к исходному коду, мы можем воспользоваться этими утилитами и узнать, что действительно происходит в программе.