Tutorial 7: Signals and Data Structures

Faculty of Engineering and Applied Science

SOFE 3950U: Operating Systems | CRN: 74171

Due: March 11th, 2024

Group 8

Daniel Amasowomwan [100787640]
daniel.amasowomwan@ontariotechu.net

Stanley Watemi [100648403]
stanley.watemi@ontariotechu.net

Fayomi Toyin [100765921]
oluwatoyin.fayomi@ontariotechu.net

Conceptual Questions.

1. What are Signals,what is their purpose?

A signal is a message that is issued by the operating system to a program or used to notify a process or thread of a situation occurring in the system.
Its purpose is to handle events just like user input, system interrupt and executing a signal handler function. This allows the programs to respond to external events dynamically.

2. Explain the following signals: SIGINT,SIGTSTP, SIGCONT, how can they be used to suspend, resume and terminate a program?
    a. SIGINT- This is the signal generated when a user keys in ctrl + c on the keyboard which terminates a program. It is sent by the operating system to interrupt a process.
    b. SIGTSTP- This instructs a process to stop. It is used to suspend a program's execution when a user keys in ctrl+z. The program is put in the background and its execution is paused.
    c. SIGCONT: This instructs the process to continue executing after it was initially paused. It is used to resume the execution of a program that has been stopped or paused by the SIGTSTP instruction.

3. Explain the following functions: kill(), waitpid() and how can they be used to terminate a process and wait until it has ended before continuing?
    a. kill()- This function can be used to terminate a process by sending it a signal like SIGKILL.
    b. waitpid(): It can be used to wait until a specific process has ended before continuing execution.
4. Explain what a linked-list(queue) is, what does FIFO mean? What are the common operations that a linked-list must have?
   A linked list is a data structure that contains nodes where each node contains data of the next node in the sequence. It can be used to implement a FIFO data structure where the first element added to the queue is the first one to be removed.
   FIFO(First-in-First-out)- The element added first in the data structure will be removed first.
   Common operations include:
    a. Enqueue(Insertion): Elements added to the end of the linked list
    b. Dequeue(Deletion): Removing elements from the front of the list
    c. Search:
5. Explain the structure of a linked list as implemented in C, explain how you would implement the operations to add and remove values from the queue?

In C, it is a data structure where each node contains two parts: data and a pointer to the next node in the sequence. The first node is the head and the last node points to NULL. The head of the list points to the first node and each subsequent node points to the next node.
In order to implement the add and remove operations, we use the Enqueue function which adds a new node with the given value to the end of the linked list and the Dequeue function which deletes the from the front of the linked list and return its value.

Application problems

```c
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>


 struct proc
{
    char name [256];
    int priority;
    int pid;
    int runtime;
};

struct queue {
    struct proc *proc;
    struct queue *next;
};

struct queue *head = NULL;

void push(struct proc *proc) {
    struct queue *new = malloc(sizeof(struct queue));

    if (new == NULL) {
        printf("Error: malloc failed\n");
        exit(1);
    }

```

```
34      new->proc = proc;
35      new->next = NULL;
36
37      if (head == NULL) {
38          head = new;
39          return;
40
41      }
42      struct queue *current = head;
43      while (current->next != NULL) {
44          current = current->next;
45      }
46      current->next = new;
47  }
48
49
50  void display() {
51      struct queue *current = head;
52      while (current != NULL) {
53          printf(" %s,  %d, %d, %d\n", current->proc->name,
54           current->proc->priority, current->proc->pid, current->proc->runtime);
55          current = current->next;
56      }
57  }
58
59  int main (){
60      FILE *file= fopen("processes.txt", "r");
61      if (file == NULL) {
62          printf("Error: file not found\n");
63          exit(1);
64      }
```

```
59  int main (){
60      FILE *file= fopen("processes.txt", "r");
61      if (file == NULL) {
62          printf("Error: file not found\n");
63          exit(1);
64      }
65      char line[256];
66      while (fgets(line, sizeof(line), file)) {
67          struct proc *proc = malloc(sizeof(struct proc));
68
69          sscanf(line, "%s %d %d %d", proc->name, &proc->priority, &proc->pid, &proc->runtime);
70          push(proc);
71
72
73      }
74      fclose(file);
75      display();
76      return 0;
77
78
79
80  }
```

```
stanley@stanley-VirtualBox:~/Tutorial7$ gcc -o question1 question1.c
stanley@stanley-VirtualBox:~/Tutorial7$ ./question1
 systemd,,  0, 0, 0
 bash,,  0, 0, 0
 vim,,  1, 0, 0
 emacs,,  3, 0, 0
 chrome,,  1, 0, 0
 chrome,,  1, 0, 0
 chrome,,  1, 0, 0
 gedit,,  2, 0, 0
 eclipse,,  2, 0, 0
 clang,,  1, 0, 0
```

2.

```c
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <string.h>

struct proc {
    char name[256];
    int priority;
    int pid;
    int runtime;
};

struct queue {
    struct proc *proc;
    struct queue *next;
};

struct queue *head = NULL;

void push(struct proc *proc) {
    struct queue *new = malloc(sizeof(struct queue));

    if (new == NULL) {
        printf("Error: malloc failed\n");
        exit(1);
    }

    new->proc = proc;
    new->next = NULL;
```

```
35
36      if (head == NULL) {
37          head = new;
38          return;
39      }
40
41      struct queue *current = head;
42      while (current->next != NULL) {
43          current = current->next;
44      }
45      current->next = new;
46  }
47
48  struct proc *pop() {
49      struct proc *proc = NULL;
50      if (head == NULL) {
51          printf("Queue is empty\n");
52      } else {
53          struct queue *temp = head;
54          head = head->next;
55          proc = temp->proc;
56          free(temp);
57      }
58      return proc;
59  }
60
61  struct proc delete_name(char *name) {
62      struct proc delete_process;
63      struct queue *previous = NULL;
64      struct queue *current = head;
65
66      while (current != NULL && strcmp(current->proc->name, name) != 0) {
67          previous = current;
68          current = current->next;
```

```
69      }
70
71      if (current != NULL) {
72          if (previous != NULL) {
73              previous->next = current->next;
74          } else {
75              head = current->next;
76          }
77          delete_process = *current->proc;
78          free(current->proc);
79          free(current);
80      } else {
81          printf("Process not found\n");
82          strcpy(delete_process.name, "NULL");
83          delete_process.priority = -1;
84          delete_process.pid = -1;
85          delete_process.runtime = -1;
86      }
87      return delete_process;
88  }
89
90  void display() {
91      struct queue *current = head;
92      while (current != NULL) {
93          printf("%s\n", current->proc->name);
94          current = current->next;
95      }
96  }
97
98  int main() {
99      FILE *file = fopen("processes.txt", "r");
100     if (file == NULL) {
101         printf("Error: file not found\n");
102         exit(1);
```

```
100     if (file == NULL) {
101         printf("Error: file not found\n");
102         exit(1);
103     }
104
105     char name[256];
106     while (fgets(name, sizeof(name), file)) {
107         struct proc *proc = malloc(sizeof(struct proc));
108         if (proc == NULL) {
109             printf("Error: malloc failed\n");
110             exit(1);
111         }
112
113         name[strcspn(name, "\n")] = '\0';
114         strcpy(proc->name, name);
115         proc->priority = rand() % 10;
116         proc->pid = -1;
117         proc->runtime = rand() % 10;
118         push(proc);
119     }
120     fclose(file);
121
122     display();
123     struct proc *popped_proc = pop();
124     if (popped_proc != NULL) {
125         printf("Popped: %s\n", popped_proc->name);
126         free(popped_proc);
127     }
128
129     display();
130     struct proc delete_process = delete_name("process1");
131     printf("Deleted: %s\n", delete_process.name);
132
133     display();
134     return 0;
```

```
stanley@stanley-VirtualBox:~/Tutorial7$ gcc -o question2 question2.c
stanley@stanley-VirtualBox:~/Tutorial7$ ./question2
systemd, 0, 1, 5
bash, 0, 1000, 8
vim, 1, 11992, 3
emacs, 3, 11993, 1
chrome, 1, 11996, 2
chrome, 1, 11997, 3
chrome, 1, 11998, 1
gedit, 2, 12235, 4
eclipse, 2, 14442, 2
clang, 1, 9223, 3
Popped: systemd, 0, 1, 5
bash, 0, 1000, 8
vim, 1, 11992, 3
emacs, 3, 11993, 1
chrome, 1, 11996, 2
chrome, 1, 11997, 3
chrome, 1, 11998, 1
gedit, 2, 12235, 4
eclipse, 2, 14442, 2
clang, 1, 9223, 3
```

```
Deleted: NULL
bash, 0, 1000, 8
vim, 1, 11992, 3
emacs, 3, 11993, 1
chrome, 1, 11996, 2
chrome, 1, 11997, 3
chrome, 1, 11998, 1
gedit, 2, 12235, 4
eclipse, 2, 14442, 2
clang, 1, 9223, 3
stanley@stanley-VirtualBox:~/Tutorial7$
```

3.

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


void child_process() {
   printf("\033[1;32m Child process is running %d\n", getpid());
   printf("Parent process id: %d\n", getppid());
   printf("child group id: %d\033[0m\n", getpgid(0));
    sleep(5);
    printf("\033[1;32m[+] Child process is exiting\033[0m\n");
}

int main() {
    pid_t pid;
    pid = fork();
    if (pid < 0) {
        printf("\033[1;31m[-] Error in fork system call\033[0m\n");
        exit(1);
    }
    if (pid == 0) {
        child_process();
    } else {
        printf("\033[1;32m[+] Parent process is running\033[0m\n");
        printf("Parent process id: %d\n", getpid());
        printf("Parent group id: %d\033[0m\n", getpgid(0));
        sleep(10);
        printf("\033[1;32m[+] Parent process is exiting\033[0m\n");
    }
    return 0;
}
```

```
stanley@stanley-VirtualBox:~/Tutorial7$ gcc -o question3 question3.c
stanley@stanley-VirtualBox:~/Tutorial7$ ./question3
[+] Parent process is running
Parent process id: 6462
Parent group id: 6462
 Child process is running 6463
Parent process id: 6462
child group id: 6462
[+] Child process is exiting
[+] Parent process is exiting
stanley@stanley-VirtualBox:~/Tutorial7$
```

4.

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

void child_process() {
   printf(" Child process is running %d\n", getpid());
   printf("Parent process id: %d\n", getppid());
   printf("child group id: %d\033[0m\n", getpgid(0));
    sleep(5);
    printf("\033[1;31m[-] Child process suspended\033[0m\n");
    raise(SIGSTOP);
    sleep(10);

    printf("\033[1;33m[+] Child process continues\033[0m\n");
}

int main (){
    pid_t pid;
    pid = fork();
    if (pid < 0) {
        printf("\033[1;31m[-] Error in fork system call\033[0m\n");
        exit(1);
    }
    if (pid == 0) {
        child_process();
    } else {
        printf("Parent process is running Chid Pid: %d\n", pid);
        sleep(10);
        printf("sending SIGCONT signal to child process\n");
        kill(pid, SIGCONT);
        waitpid(pid, NULL, 0);
```

```
stanley@stanley-VirtualBox:~/Tutorial7$ gcc -o question4 question4.c
stanley@stanley-VirtualBox:~/Tutorial7$ ./question4
Parent process is running Chid Pid: 7978
 Child process is running 7978
Parent process id: 7977
child group id: 7977
[-] Child process suspended
sending SIGCONT signal to child process
[+] Child process continues
stanley@stanley-VirtualBox:~/Tutorial7$
```

5.

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

typedef struct Process {
    pid_t pid;
    char name[1000];
    int priority;
    int runtime;
    struct Process *next;
} Process;

typedef struct Queue {
    Process *head;
    Process *tail;
} Queue;

void initQueue(Queue *queue) {
    queue->head = NULL;
    queue->tail = NULL;
}
int emptyQueue(Queue *queue) {
    Process *p = queue->head;
    while (p != NULL) {
        Process *temp = p;
        p = p->next;
        free(temp);
    }
    queue->head = NULL;
    queue->tail = NULL;
}
void enqueue(Queue *queue,char *name,int priority,int runtime) {
```

```c
void enqueue(Queue *queue,char *name,int priority,int runtime) {
    Process *process = (Process *)malloc(sizeof(Process));
    strcpy(process->name, name);
    process->priority = priority;
    process->runtime = runtime;
    process -> pid = -1;
    process->next = NULL;

    if (queue->head == NULL) {
        queue->head = process;
        queue->tail = process;
    } else {
        queue->tail->next = process;
        queue->tail = process;
    }
}
void deleteName(Queue *queue,char *name) {
    Process *p = queue->head;
    Process *prev = NULL;
    while (p != NULL) {
        if (strcmp(p->name, name) == 0) {
            if (prev == NULL) {
                queue->head = p->next;
            } else {
                prev->next = p->next;
            }
            if (p == queue->tail) {
                queue->tail = prev;
            }
            free(p);
            return;
```

```
67          }
68          prev = p;
69          p = p->next;
70      }
71  }
72  void executeProcess(Process *process) {
73      pid_t pid = fork();
74      if (pid == 0) {
75          printf("Executing %s\n", process->name);
76          execlp("./process", "./process", process->name,NULL);
77          perror("execlp");
78          exit(0);
79      } else {
80          process->pid = pid;
81          sleep(process->runtime);
82          printf("Terminate process:%s(PID:%d)\n", process->name, process->pid);
83          kill(pid, SIGINT);
84          waitpid(pid, NULL, 0);
85          printf("Name: %s, Priority: %d, Runtime: %d\n", process->name, process->priority, process->runtime);
86      }
87  }
88  int main() {
89      Queue queue;
90      initQueue(&queue);
91     FILE *file = fopen("processes_q5.txt", "r");
92      if (file == NULL) {
93          printf("File not found\n");
94          return 1;
95      }
96      char name[1000];
97      int priority;
98      int runtime;
99      while (fscanf(file, "%s %d %d", name, &priority, &runtime) != EOF) {
100         enqueue(&queue, name, priority, runtime);
```

```
100         enqueue(&queue, name, priority, runtime);
101     }
102     fclose(file);
103     Process *p = queue.head;
104     while (p != NULL) {
105         executeProcess(p);
106         deleteName(&queue, p->name);
107         p = p->next;
108     }
109     while (!emptyQueue(&queue)) {
110         Process *p = queue.head;
111         executeProcess(p);
112         deleteName(&queue, p->name);
113
114     }
115     return 0;
116
117 }
```

```
stanley@stanley-VirtualBox:~/Tutorial7$ gcc -o question5 question5.c
stanley@stanley-VirtualBox:~/Tutorial7$ ./question5
Terminate process:systemd,(PID:9112)
Executing systemd,
Name: systemd,, Priority: 0, Runtime: 0
Terminate process:,(PID:9113)
Executing ,
Name: ,, Priority: 5, Runtime: 0
Terminate process:bash,(PID:9114)
Name: bash,, Priority: 0, Runtime: 0
Executing ,
Terminate process:,(PID:9115)
Name: ,, Priority: 8, Runtime: 0
Terminate process:vim,(PID:9116)
Name: vim,, Priority: 1, Runtime: 0
Terminate process:,(PID:9117)
Name: ,, Priority: 3, Runtime: 0
Terminate process:emacs,(PID:9118)
Name: emacs,, Priority: 3, Runtime: 0
Terminate process:,(PID:9119)
Name: ,, Priority: 1, Runtime: 0
Terminate process:chrome,(PID:9120)
Name: chrome,, Priority: 1, Runtime: 0
Terminate process:,(PID:9121)
```