COSC264 Major Assignment

Jingwei Li #39923031 (50%)

Juanpeng Qiu #35635713 (50%)

1. Processes need access to resources in reasonable order. Suppose a process holds resource A and requests resource B, and at the same time another process holds B and requests A, both are blocked and remain so.

Deadlocks occur when processes are granted exclusive access to resources such as printers, tape drives and tables.

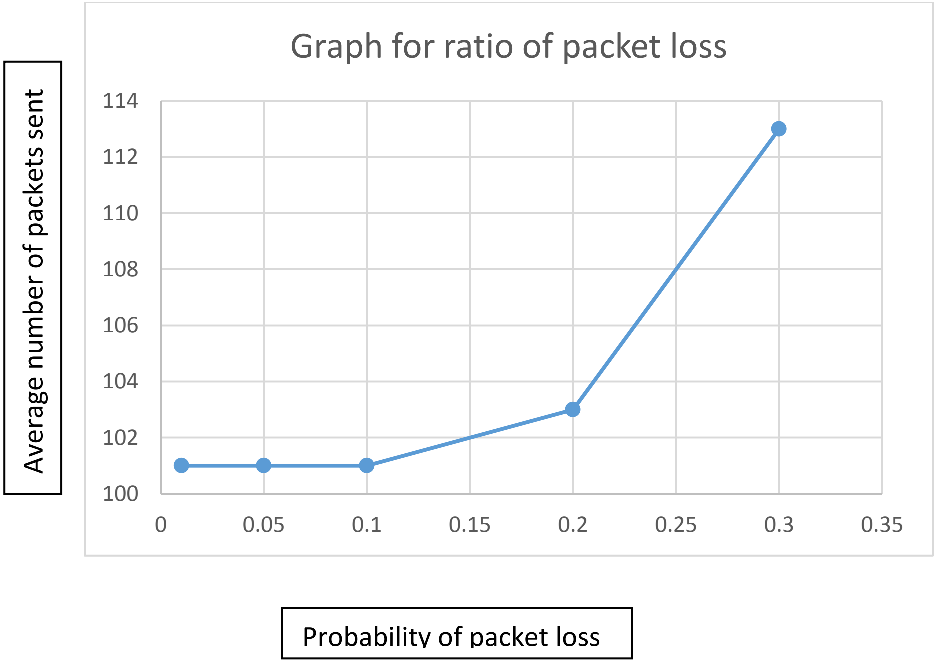
In this case, the resources are files and sockets. For example, if the sender or receiver program requests a file when another process holds the file, the deadlock occurs and that packets are lost.

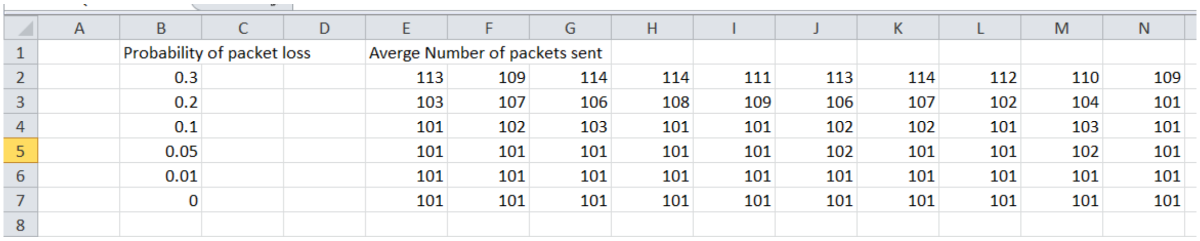
2. The magicno can work as a identification or checksum, because the packets in the transmission can get lost, get modified or being tampered with.

3. Select() is a system call in Linux. It examines the status of file descriptors of open I/O channels. It waits for input on any one of the “in” sockets. The control is not returned to the program until at least one byte of data is ready to read. It saves CPU time.

4. Using diff command (diff FILE1 FILE2) to display line by line difference between the receivers copy and the transmitters copy. It examines both files and tells what changes need to be made for these files to match.

5.





	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		Probability of packet loss			Averge Number of packets sent									
2		0.3			113	109	114	114	111	113	114	112	110	109
3		0.2			103	107	106	108	109	106	107	102	104	101
4		0.1			101	102	103	101	101	102	102	101	103	101
5		0.05			101	101	101	101	101	102	101	101	102	101
6		0.01			101	101	101	101	101	101	101	101	101	101
7		0			101	101	101	101	101	101	101	101	101	101
8														

In conclusion, the graph happened to be an increasing curve as the probability of packet loss increase, the number of packets sent increases. Therefore when the probability is 0 no packets loss happened and we did not seem to find any difference for the number close to 0 too.

6. n*(1/(1-P))^2. The probability of failure from sender to receiver is P, (1 – P) then is for the probability of successfully delivered data. 1/(1 – P) is for how many times needed to be sent in order for receiver to receive. If both data packet and acknowledgement packet were successfully sent, (1/(1 – P))^2. And n*(1/(1 – P))^2 means n number of packets had been successfully sent.

Source Code

channel.py

```
from sys import *
from socket import *
from struct import *
from os.path import *
from select import *
from packet import *
from random import *

host = '127.0.0.1'

def in_range(ports):
    for port in ports:
        if port in range(1024, 64000):
            return True
        else:
            return False

def main(argv):
    for i in range(len(argv)-1):
        argv[i] = int(argv[i])
    if in_range(argv[0:-1]):
        if float(argv[-1]) < 1 and float(argv[-1]) >= 0:
            cs_in = argv[0]
            cs_out = argv[1]
            cr_in = argv[2]
            cr_out = argv[3]
```

```
s_in = argv[4]
r_in = argv[5]
p = 1 - float(argv[6])

cs_in_s = socket(AF_INET, SOCK_DGRAM)
try:
    cs_in_s.bind((host, cs_in))
    print("Channel s in socket bind Successfully")
except: print("Channel s in socket bind Unsuccessfully")
cs_out_s = socket(AF_INET, SOCK_DGRAM)
try:
    cs_out_s.connect((host, s_in))
    print("Channel s out socket connect Successfully")
except: print("Channel s out socket connect Unsuccessfully")
cr_in_s = socket(AF_INET, SOCK_DGRAM)
try:
    cr_in_s.bind((host, cr_in))
    print("Channel r in socket bind Successfully")
except: print("Channel r in socket bind Unsuccessfully")
cr_out_s = socket(AF_INET, SOCK_DGRAM)
try:
    cr_out_s.connect((host, r_in))
    print("Channel r out socket connect Successfully")
except: print("Channel r out socket connect Successfully")

magic_no = 0x497E
inputs = []
outputs = []
inputs.append(cs_in_s)
inputs.append(cr_in_s)
```

```
while True:
    readable,writable,exceptional = select(inputs,outputs,inputs)

    if not readable:
        print("waiting for inputs")

    for s in readable:
        if s is cs_in_s:
            buffer = s.recv(550)
            head = unpack("iiii",buffer[0:16])

            if head[0] != magic_no:
                print("Wrong packet\n")
                break
            else:
                u = uniform(0, 1)
                if u < p:
                    print("A packet have been dropped\n")
                else:
                    size = head[3]
                    if size == 0:
                        head = pack("iiii", head[0], head[1], head[2], head[3])
                        try:
                            cr_out_s.send(head)
                            print("A packet is sending to receiver\n")
                        except: print("A packet cannot send to sender\n")

                    elif size > 0:
                        data = unpack("iiii"+str(size)+"s", buffer)
```

```
                                    buffer = pack("iiii"+ str(size)+ "s",head[0],head[1],head[2],head[3],data[4])
                                    cr_out_s.send(buffer)
                                    print("A packet is sending to receiver\n")

                    elif s is cr_in_s:
                        buffer = s.recv(550)
                        head = unpack("iiii",buffer[0:16])

                        if head[0] != magic_no:
                            print("Wrong packet\n")
                            break
                        else:
                            u = uniform(0, 1)
                            if u < p:
                                print("A packet have been dropped\n")
                            else:
                                head = pack("iiii",head[0],head[1],head[2],head[3])
                                try:
                                    cs_out_s.send(head)
                                    print("A packet is sending to sender\n")
                                except: print("A packet cannot send to sender\n")
        else:
            print("invalid loss probabilities")
            exit(0)
    else:
        print("invalid port number")
        exit(0)
if __name__ == "__main__":
    main(argv[1:])
```

receiver.py

```
from sys import *
from socket import *
from struct import *
from os.path import *
from select import *
from packet import *

host = '127.0.0.1'

def in_range(ports):
    for port in ports:
        if port in range(1024, 64000):
            return True
        else:
            return False

def main(argv):
    for i in range(len(argv)-1):
        argv[i] = int(argv[i])

    if in_range(argv[0:3]):
        if exists(argv[3]):
            r_in = argv[0]
            r_out = argv[1]
            cr_in = argv[2]

            r_in_s = socket(AF_INET, SOCK_DGRAM)
            try:
```

```
    r_in_s.bind((host, r_in))
    print("Receiver in socket bind Successfully")
except: print("Receiver in socket bind Unsuccessfully")
r_out_s = socket(AF_INET, SOCK_DGRAM)
try:
    r_out_s.connect((host, cr_in))
    print("Receiver out socket connect Successfully")
except: print("Receiver out socket connect unsuccessfully")

fo = open(argv[3],"wb")
expected = 0
magic_no = 0x497E
buffer = ()
inputs = [r_in_s]
outputs = []

while True:
    readable,writable,exceptional = select(inputs,outputs,inputs)
    for s in readable:
        if s is r_in_s:
            buffer = s.recv(550)
            head = unpack("iiii",buffer[0:16])
            if head[0] != magic_no or head[1] != 0:
                print("stop processing.\n")
            else:
                if head[2] != expected:
                    a_packet = pack("iiii", magic_no, 1, head[2], 0)
                    r_out_s.send(a_packet)
                    print("stop processing.\n")
                else:
```

```
                                a_packet = pack("iiii", magic_no, 1, head[2], 0)
                                r_out_s.send(a_packet)
                                expected = 1 - expected
                                size = head[3]
                                if size > 0:
                                    data = unpack(str(size)+"s",buffer[16:])
                                    fo.write(data[0])
                                elif size == 0:
                                    r_in_s.close()
                                    r_out_s.close()
                                    fo.close()
                                    exit(0)
        else:
            print("file name error")
            exit(0)
    else:
        print("invalid port number")
        exit(0)


if __name__ == "__main__":
    main(argv[1:])
```

sender.py

```
from sys import *
from socket import *
from struct import *
from os.path import *
from select import *
from packet import *

host = '127.0.0.1'

def in_range(ports):
    for port in ports:
        if port in range(1024, 64000):
            return True
        else:
            return False

def main(argv):
    for i in range(len(argv)-1):
        argv[i] = int(argv[i])
    if in_range(argv[0:3]):
        if exists(argv[3]):
            s_in = argv[0]
            s_out = argv[1]
            cs_in = argv[2]

            s_in_s = socket(AF_INET, SOCK_DGRAM)
            try:
                s_in_s.bind((host, s_in))
                print("Sender in socket bind Successfully")
            except: print("Sender in socket bind Unsuccessfully")
            s_out_s = socket(AF_INET, SOCK_DGRAM)
            try:
                s_out_s.connect((host, cs_in))
                print("Sender out socket connect Successfully")
            except: print("Sender out socket connect Unsuccessfully")

            fo = open(argv[3],"rb")
            nxt = 0
            exitFlag = False
            buffer = ()
            counter = 0
            magic_no = 0x497E
            inputs = [s_in_s]
            outputs = []

            while not exitFlag:
                data = fo.read(512)
                size = len(data)
                if size == 0:
                    buffer = pack("iiii",magic_no,0,nxt,size)
                    exitFlag = True
                elif size > 0:
                    buffer = pack("iiii"+str(size)+"s",magic_no,0,nxt,size,data)

                while True:
                    s_out_s.send(buffer)
                    counter += 1
                    print("Sending a packet\n")
```

```
                readable,writable,exceptional = select(inputs,outputs,inputs,1)


                if not readable:
                    print("time out")
                    break
                for s in readable:
                    if s is s_in_s:
                        a_packet = unpack("iiii",s.recv(16))
                        print("Received a packet.")
                        if a_packet[0] != magic_no or a_packet[1] != 1 or a_packet[2] != nxt or a_packet[3] != 0:
                            print("Re-transmitting a packet\n")
                            break
                        elif a_packet[2] == nxt:
                            nxt = 1 - nxt
                            if exitFlag:
                                print("packets in total {}".format(counter))
                                s_in_s.close()
                                s_out_s.close()
                                fo.close()
                                exit(0)
                            else:
                                break


            print("packets in total {}".format(counter))
    else:
        print("file name error")
        exit(0)
else:
    print("invalid port number")
```

```
        exit(0)


if __name__ == "__main__":
    main(argv[1:])
```