**ROLL NO: 13BCE109**

**TOPIC: MEMORY MANAGEMENT USING LINKED LISTS**

* **SOURCE CODE:**

# coding: utf-8

# In[1]:

import sys

from tkMessageBox import \*

from Tkinter import \*

# In[2]:

class Process(object):

def \_\_init\_\_(self, name, size):

self.\_size = size

self.\_name = name

def get\_size(self):

return self.\_size

def get\_name(self):

return self.\_name

# In[3]:

class Segment(object):

def \_\_init\_\_(self, is\_process, start\_address, process=None, size=None, previous=None, nex=None):

if((is\_process==False and (process!=None or size==None)) or (is\_process==True and (process==None or size!=None))):

print "Error!!! in initializing segment",is\_process,process,size

return

self.\_type = is\_process

self.\_process = process

self.\_pre = previous

self.\_next = nex

self.\_start\_address = start\_address

if is\_process:

self.\_size = process.get\_size()

else:

self.\_size = size

def make\_hole(self, size=None):

if size == None:

self.\_size = self.\_process.get\_size()

elif size < self.\_size:

print "new hole size cant be smaller",size,self.\_size

return

else:

self.\_size = size

self.\_type = False

self.\_process = None

def get\_previous(self):

return self.\_pre

def get\_next(self):

return self.\_next

def get\_process(self):

return self.\_process

def get\_start\_address(self):

return self.\_start\_address

def get\_size(self):

return self.\_size

def set\_previous(self, segment):

self.\_pre = segment

def set\_next(self, segment):

self.\_next = segment

def set\_size(self,size):

self.\_size = size

def set\_start\_address(self,start\_address):

self.\_start\_address = start\_address

def make\_process(self, process):

if(self.\_type):

print "Already a process"

return

self.\_process = process

self.\_type = True

self.\_size = process.get\_size()

def is\_process(self):

return self.\_type

# In[4]:

class Memory(object):

def \_\_init\_\_(self,size,algorithm=1):

self.\_size = size

self.\_head = Segment(False,start\_address=0,size=self.\_size)

self.\_algo = algorithm

self.\_list = []

print 'Tot Size :',size

print 'Algo :',algorithm

if algorithm==2:

self.\_nex = self.\_head

self.display2()

#self.diplay(1)

def remove\_process(self, name):

print '#########################'

print 'Removing:',name

if name not in self.\_list:

print 'Process does not exist'

return

self.\_list.remove(name)

seg = self.\_head

while seg != None:

if seg.is\_process():

if seg.get\_process().get\_name()==name:

break

seg = seg.get\_next()

if seg == None:

print "No such process"

return

pre = seg.get\_previous()

nex = seg.get\_next()

if pre==None and nex==None:

seg.make\_hole()

elif pre==None:

if nex.is\_process():

seg.make\_hole()

else:

seg.make\_hole(seg.get\_size()+nex.get\_size())

seg.set\_next(nex.get\_next())

if nex.get\_next():

nex.get\_next().set\_previous(seg)

elif nex==None:

if pre.is\_process():

seg.make\_hole()

else:

pre.set\_size(pre.get\_size()+seg.getsize())

pre.set\_next(None)

elif pre.is\_process() and nex.is\_process():

seg.make\_hole()

elif pre.is\_process() and not nex.is\_process():

seg.set\_next(nex.get\_next())

seg.make\_hole(seg.get\_size()+nex.get\_size())

if nex.get\_next():

nex.get\_next().set\_previous(seg)

elif not pre.is\_process() and nex.is\_process():

pre.set\_next(nex)

pre.set\_size(pre.get\_size()+seg.get\_size())

nex.set\_previous(pre)

elif not pre.is\_process() and not nex.is\_process():

pre.set\_next(nex.get\_next())

pre.set\_size(pre.get\_size()+seg.get\_size()+nex.get\_size())

if nex.get\_next():

nex.get\_next().set\_previous(pre)

self.display()

def add\_process(self,name,size):

print '#########################'

print 'Adding:',name

if name in self.\_list:

showerror('Unable to add process','Process with the same name already exists')

return

self.\_list.append(name)

seg = self.\_head

algo = self.\_algo

process = Process(name,size)

#first-fit

if algo==1:

while seg!=None:

if not seg.is\_process() and seg.get\_size()>=process.get\_size():

break

seg = seg.get\_next()

#next-fit

elif algo==2:

seg = self.\_nex

while seg!=None:

if not seg.is\_process() and seg.get\_size()>=process.get\_size():

break

seg = seg.get\_next()

if not seg:

seg = self.\_head

while seg!=None:

if not seg.is\_process() and seg.get\_size()>=process.get\_size():

break

seg = seg.get\_next()

if seg:

self.\_nex = seg

#best-fit

elif algo==3:

mini = sys.maxint

best = None

while seg!=None:

if not seg.is\_process() and seg.get\_size()>=process.get\_size() and (seg.get\_size()-process.get\_size())<mini:

best = seg

mini = seg.get\_size()-process.get\_size()

seg = seg.get\_next()

seg = best

#worst-fit

elif algo==4:

maxi = -1

worst = None

while seg!=None:

if not seg.is\_process() and seg.get\_size()>=process.get\_size() and (seg.get\_size()-process.get\_size())>maxi:

worst = seg

maxi = seg.get\_size()-process.get\_size()

seg = seg.get\_next()

seg = worst

if seg==None:

showerror('Unable to add process','Process too big')

self.\_list.remove(name)

return

n = seg.get\_next()

if seg.get\_size() > size:

n = Segment(False, seg.get\_start\_address()+size, previous=seg, nex=seg.get\_next(), size=seg.get\_size()-size)

seg.make\_process(process)

seg.set\_next(n)

self.display()

def add(self):

a = self.\_new\_process\_name.get()

b = self.\_new\_process\_size.get()

if a=='' or b=='':

showerror('Invalid input','Please fill both process name and process size')

return

b= int(b)

if b<1:

showerror('Invalid input','Please fill correct process size')

return

self.add\_process(a,b)

def remove(self):

a = self.\_rem\_process\_name.get()

if a=='':

showerror('Invalid input','Please select the name of the process to be removed')

print 'To be removed:',a

self.remove\_process(a)

def display1(self,init=None):

seg = self.\_head

while seg!=None:

if seg.is\_process():

print 'Process:',seg.get\_process().get\_name(),seg.get\_process().get\_size(),seg.get\_size(),seg.get\_start\_address()

else:

print 'Hole:',seg.get\_size(),seg.get\_start\_address()

seg = seg.get\_next()

print self.\_list

print '#########################'

def display2(self):

out = Tk()

#

frame\_up = Frame(out)

frame\_mid = Frame(out)

frame\_down = Frame(out)

#

add\_process\_label = Label(frame\_up, text="ADD PROCESS",bg='black',fg='white')

add\_process\_label.pack(fill=X)

add\_process\_frame = Frame(frame\_up)

add\_process\_label = Label(add\_process\_frame, text="Process Name : ")

add\_process\_label.grid(row=0,column=0)

add\_process\_name = Entry(add\_process\_frame)

add\_process\_name.grid(row=0,column=1)

add\_process\_label = Label(add\_process\_frame, text="Process Size : ")

add\_process\_label.grid(row=1,column=0)

add\_process\_size = Entry(add\_process\_frame)

add\_process\_size.grid(row=1,column=1)

add\_process\_frame.pack(fill=X,pady=20)

add\_button = Button(frame\_up,text='Add',command=self.add)

add\_button.pack(fill=X)

self.\_new\_process\_name = add\_process\_name

self.\_new\_process\_size = add\_process\_size

#

memory\_label = Label(frame\_down, text="MEMORY",bg='black',fg='white')

memory\_label.pack(fill=X)

#

frame\_up.pack(fill=X)

frame\_mid.pack(fill=X,pady=30)

frame\_down.pack(fill=X,pady=30)

self.\_frame\_rem = frame\_mid

self.\_frame\_rem\_temp = None

self.\_frame\_process = frame\_down

self.\_frame\_process\_temp = None

self.\_rem\_list = None

#

self.\_root = out

out.mainloop()

self.display()

def display(self,init=None):

#

if self.\_frame\_process\_temp:

self.\_frame\_process\_temp.pack\_forget()

frame\_process = self.\_frame\_process

big\_frame = Frame(frame\_process)

seg = self.\_head

while seg!=None:

nframe = None

if seg.is\_process():

nframe = Frame(big\_frame,bg='yellow',bd='2')

name = seg.get\_process().get\_name()

name\_label = Label(nframe, text=name,bg='black',fg='white')

name\_label.pack(fill=X)

size = seg.get\_size()

size\_label = Label(nframe, text=size,bg='yellow')

size\_label.pack(fill=X)

address = seg.get\_start\_address()

address\_label = Label(nframe, text=address,bg='yellow')

address\_label.pack()

else:

nframe = Frame(big\_frame,bd='2')

name\_label = Label(nframe, text='Hole',bg='black',fg='white')

name\_label.pack(fill=X)

size = seg.get\_size()

size\_label = Label(nframe, text=size)

size\_label.pack(fill=X)

address = seg.get\_start\_address()

address\_label = Label(nframe, text=address)

address\_label.pack()

nframe.pack(side=LEFT,fill=Y)

seg = seg.get\_next()

big\_frame.pack(pady=10)

self.\_frame\_process\_temp = big\_frame

#

if self.\_frame\_rem\_temp:

self.\_frame\_rem\_temp.pack\_forget()

out = self.\_root

l = self.\_list

if len(l)>0:

big\_frame = Frame(self.\_frame\_rem)

rem\_process\_label = Label(big\_frame, text="REMOVE PROCESS",bg='black',fg='white')

rem\_process\_label.pack(fill=X)

rem = StringVar(out)

rem\_list = OptionMenu(big\_frame, rem, \*l)

rem\_list.pack(pady=10)

rem\_button = Button(big\_frame,text='Remove',command=self.remove)

rem\_button.pack(fill=X)

self.\_rem = rem

big\_frame.pack(fill=X)

self.\_frame\_rem\_temp = big\_frame

self.\_rem\_process\_name = rem

else:

self.\_frame\_rem\_temp = None

self.display1()

# In[5]:

from Tkinter import \*

intro = Tk()

title = Label(intro,text='MEMORY MANAGEMENT',bg='black',fg='white',font='Times 32',padx=20)

title.pack()

frame\_1 = Frame(intro)

frame\_1.pack()

frame\_size = Frame(frame\_1,padx=20,pady=20)

frame\_size.pack(fill=X)

size\_label = Label(frame\_size, text="Memory Size : ")

size\_label.pack(side=LEFT)

size = Entry(frame\_size)

size.pack(side=LEFT)

frame\_algos = Frame(frame\_1,padx=20,pady=20)

frame\_algos.pack(fill=X)

name\_label = Label(frame\_algos, text="Algorithm : ")

name\_label.pack(side=LEFT)

algo\_but = IntVar()

algo\_but.set(1)

algos = [('First-fit',1),('Next-fit',2),('Best-fit',3),('Worst-fit',4)]

frame\_opts = Frame(frame\_algos)

frame\_opts.pack(side=LEFT)

for txt,val in algos:

Radiobutton(frame\_opts, text=txt, value=val, variable=algo\_but, padx=20).pack(anchor=W)

def func():

algo = int(algo\_but.get())

sizey = int(size.get())

intro.destroy()

m = Memory(sizey,algo)

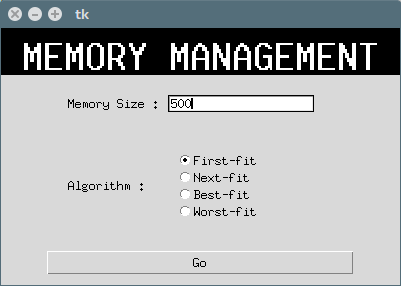
go = Button(frame\_1,text='Go',command=func,width=40)

go.pack(pady=10)

intro.mainloop()

# In[ ]:

* **SAMPLE INPUT:**



* **SAMPLE OUTPUT:**

