"""

Author: Nicholas Atkins Due Date: 3/4/2018

Purpose: To demonstrate the best fit algorithm with dynamic partitioned memory

Problems: There are a lot of if and else statements making this very hard to understand

just by reading the code, I hope the comments spread throughout the code will help understand the

thought process behind it

How to run the program:

1. Open the file into a python compiler

2. If you wish you can test the test as well, just run it like a normal .py file

3. Enter an int value greater than 20

4. Hold enter until you are satisfied

"""

# Having this module made making the random selections easier

from random import \*

def random\_job\_number():

"""

This will take create a string job integer between two numbers

:return: a string

"""

x = randint(1, 10) # controls the number of different jobs for the program to run

return 'job' + str(x)

def random\_size():

"""

This will return a random integer between

:return: an integer

"""

return randint(1, 100) # You can fiddle with this portion, this controls the max and min size of the jobs

def in\_or\_out():

"""

creates a string in or out

:return: a string

"""

x = randint(1,2) # these are set so don't change them

if x == 1:

return 'in'

else:

return 'out'

def print\_memory(listed):

""""

prints the job in the memory and the free space as well

:returns an integer

"""

contained = []

start = 0

for x in range(1, len(listed)):

if x == 1:

start = x-1

elif listed[x-1] != listed[x]:

print(listed[x-1], ' ', start, ' ', x-1)

contained.append(listed[x-1])

start = x

if(x == len(listed)-1 and listed[x] != listed[x-1]): # This had to be added, it wouldn't print last item

print(listed[x], ' ', start, ' ', x)

contained.append(listed[x])

if(x == len(listed)-1 and (listed[x] not in contained or listed[x] is None)): # Not sure if this is

print(listed[x], ' ', start, ' ', x) # Necessary however it may catch something I haven't seen

contained.append(listed[x])

return contained

def count\_none(listed): # Just a fancy tool to let you know that there are extra spaces leftover total

"""

Counts the number of free spaces left

:return: an integer

"""

count = 0

for i in range(0, len(listed)):

if listed[i] is None:

count += 1

return count

def main():

job\_dict = {} # Where the jobs are stored post allocation

max\_size = 19 # my OS takes 20 spaces therefore it can't be smaller than the OS

while(max\_size < 20):

max\_size = int(input("Enter the size of the the memory: ")) # Anything that isn't an int will break this

# would had used a try statement however, all nighters suck

memory\_list = [None] \* max\_size

for x in range(0,20):

memory\_list[x] = 'OS'

again = True

while again:

op = in\_or\_out()

if op == 'in': # This is the in section of the code

"""

Job in will check the job dict for the job, if not will

attempt to allocate space for the job if there is room otherwise will toss the job

"""

print('In was selected')

job = random\_job\_number()

size = random\_size()

found = job\_dict.get(job)

if found is None:

empty\_space = []

count = 0

start = 0

"""

A lot of time was spent here trying to find the problem with

the None track and markers

"""

for y in range(0, len(memory\_list)):

if count == 0 and memory\_list[y] is None:

start = y

count = 1

if (count == 1 and memory\_list[y] is not None) or (y == len(memory\_list)-1 and

memory\_list[len(memory\_list)-1] is None):

empty\_space.append([start, y])

start = y

count = 0

remainder = []

for y in range(0, len(empty\_space)):

space = int(empty\_space[y][1]) - int(empty\_space[y][0])

remainder.append(space-size)

index = -1

for y in range(0, len(remainder)):

if index < 0: # Added this to make sure we never get a remainder below 0

if remainder[y] >= 0:

index = y

else:

if 0 <= remainder[y] < remainder[index]:

index = y

if index == -1: # Meaning there was not enough space for allocation

print("There is not enough space for ", job)

else: # The job has enough space to be placed

print('Allocating space for ', job)

for i in range(0, size):

memory\_list[empty\_space[index][0] + i] = job

job\_dict[job] = [empty\_space[index][0], empty\_space[index][0]+size-1]

else:

startnfinish = job\_dict.get(job) # checking the dictionary set up at the start to see if it is included

print('it is in the dict file')

blank = True

print(startnfinish)

for x in range(startnfinish[0], startnfinish[1]):

if memory\_list[x] is not None:

blank = False # if something is in the spot allocated to it in the dict, it will cause a false

if blank:

for x in range(startnfinish[0], startnfinish[1]):

memory\_list[x] = job

else:

print('There is something occupying the space for that job')

else: # This is the out section of the code

"""

Job out will see if there is a process in memory list with that name and remove it

"""

job = random\_job\_number()

print('Out was selected')

if job in memory\_list: # if the job is found it will remove the job from the space it is in

for x in range(0, len(memory\_list)):

if memory\_list[x] == job:

memory\_list[x] = None

print('Cleared ', job)

else: # The job was not found in the memory so it wont remove it

print('The job ', job, 'is not in memory')

# prints off the full list minus the Nones at the end, for some reason they didn't want to print

cookie = print\_memory(memory\_list)

print(cookie, 'There are ', count\_none(memory\_list), 'blank spaces\n\n') # blank meaning spares

# Holding down enter will give the same result as typing yes, it is intended

response = str(input("Again? Yes or No: "))

if response == 'No' or response == 'no':

again = False

else:

again = True

print('\n')

if \_\_name\_\_ == '\_\_main\_\_':

main()