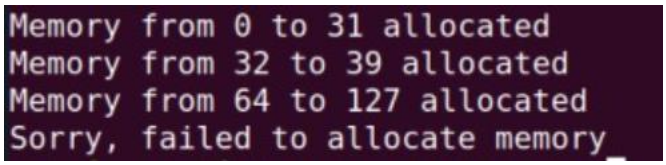
OS prac end-sem

description

AU1940121

Jinil Chandarana

2) Buddy algorithm:



In buddy algorithm, the memory is used efficiently by spiling the available space repeatedly into halves. This consecutive half split spaces are called buddies of each other. For instance, if we have a memory of 1024 GB, it is split into two 512GB spaces. name the space say, 512A and 512B. These 512A and 512B are called buddies. It keeps splitting until it chooses the smallest possible block that can contain the file with no or minimum internal fragmentation.

Allocation process;

As discussed, the memory is divided into equal parts until it is small enough to just fit the required data into it. Taking an example: lets say we have 4 different data files

A: 64GB

B:128GB

C:225GB

D:256GB

and the system have a total of 1024GB size.

1024GB

We 1st check if the memory is big enough that after splitting it can contain a file. It will have 512 GB size so it can fit the files.

512GB	512GB

We again keep dividing the lower part of memory into 2 equal parts.

512GB	256	256

512GB	256	128	64	64

We need not divide further as we have reached the size division for the smallest file i.e A of 64gb. here we store A in the right most block.

512GB	256	128	64	A(64)

Next available is flies are stored as below.

512GB	C (256) 31	B (128) 0	64	A(64) 0

[note: size in () is the size of block, size in red fonts in the remaining (fragments) in the block]

256	D (256) 0	C (256) 31	B (128) 0	64	A(64) 0

As we can see all the files except C has a right fit in the block. The file C is 225gb but the size of the block is 256. The 256-225 = 31Gb is the fragmentation we obtained.

Deallocation:

Now, for deallocation of a block, it will first check its buddy. If the buddy is not used (is empty) then and then only it will merge back together. Once merged we will have the original buddies. E.g. We can remove file A and mere the block with it's buddy to form a 128gb block and can fit a new file say E having 200 GB size (this will have fragment of 8gb).

3) producer consumer prob.

The producer and consumer problem is a synchronization problem. The fixed size of buffer is initialized. Then the producer produces an item and sends it to the buffer and stores it there. The consumer in the other hand removes the item from the buffer and uses it. One or more producers are generating data and placing into buffer. Only one producer or consumer may access the buffer at any one time. When the buffer is full, producer cannot enter data into buffer, and consumer can't remove data from the empty buffer.

==========

Problems can be solved by:

Manage shared memory access

Checking for if buffer is full

Checking for if buffer is empty

As we are accessing the same buffer from producer and consumer, we should stop the race condition i.e. if one thread is incrementing count. The other thread should not decrement it. To do that we use MUTEX. initializing and destroying mutex for each function i.e producer and consumer along with mutex lock and unlock in the code.

Now to check the full and empty buffer which might lead to skip or ignore an important production and consumption (request) if the buffer is full or empty, we use 2 semaphores. 1 for full buffer and 2nd for empty. If the buffer is full, we use sem_wait to wait for the slot to be empty and then signal it when a free space is available. Thus, semaphore keeps treak of full and empty space and sem_wait waits till at least one slot is empty when producing and al least one slot is filled then consuming. I have also initialized an int i so than we get producer in a fixed time interval to reduce the problem that producer do not produce much more than the consumer can consume.

```
Got 22
Got 92
Got 21
Got 57
Got 48
Got 68
Got 51
Got 78
Got 91
Got 35
Got 41
Got 78
Got 11
Got 1
Got 79
Got 71
Got 33
Got 77
Got 78
Got 98
Got 69
Got 22
Got 63
Got 9
Got 31
Got 49
Got 94
Got 34
```