Operating system -

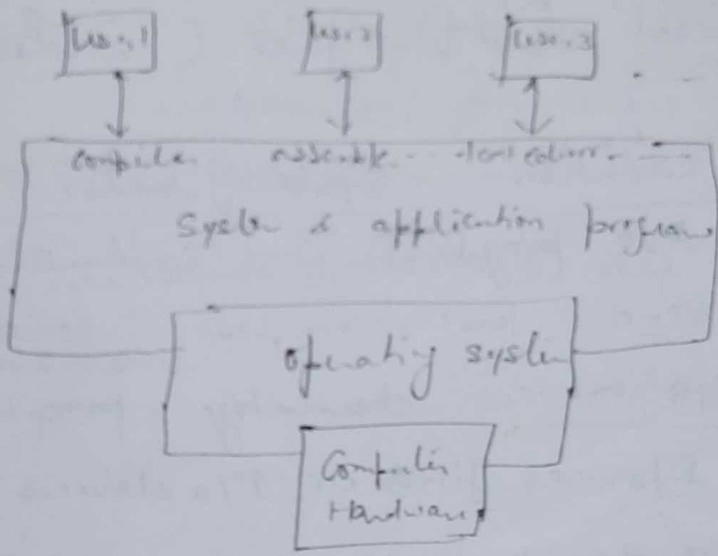
① it manages computer H/W.

② it provides convenient & efficient environment to the user

③ it works as resource allocator to application programmes

④ it works as mediator between comp. H/W & user. ( Provides Interface to user)

⑤ it is also known as control program that manages the execution of user programs to prevent errors and improper use of computer




Manager    Controller    resource allocator

Operating system services - operating system provides services for the convenience of the programmer, to make the programming task easier.

1. User Interface -

(a) Command line interface (MS-DOS)

(b) Batch Interface

(Commands & directives to control these command are entered into files & these files are executed)

(c) Graphical User Interface (windows 98 ...)

2. Program execution - System must be able to load the program in memory & run this program.

3. I/O operations - running program may require I/O (files or I/O devices). for efficiency and protection, user ~~cannot~~ can not control I/O devices directly. So O.S must provide means to do I/O.

4. File system manipulation - all the file operations must be supported or provided by O.S. like create, delete, read write, search, list files & deny or allow access of files.

5. Communications - Commⁿ within processes to share information must be supported

6. Error detection - ① H/w error (device failure)
② user programe error (overflow)

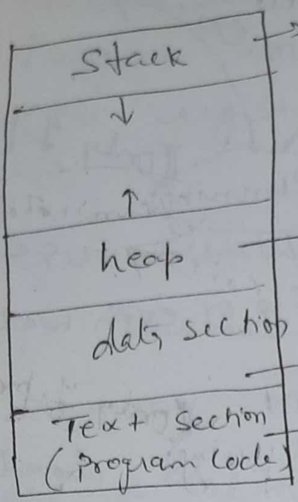
For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

For efficiency of system, services are -

1. Resource allocation - optimized resource allocation
2. Accounting - which program consuming which resource for how much time
3. Protection and security -
system resource accesses are controlled like memory

# Process

**Process** → is a program in execution.
→ it is the unit of work.



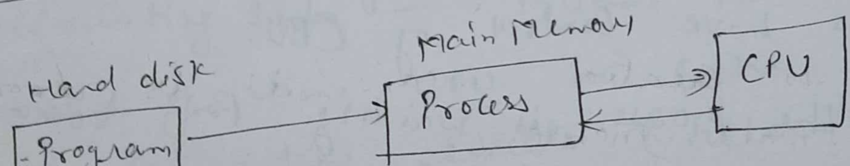

→ to hold temporary data (local var, return add, Parameter etc.)

→ Memory that is dynamically allocated during process run time.

→ Global Variables

→ Program Code + Program Counter + Processor register Contents.

## Program Vs process -




Passive entity — active entity

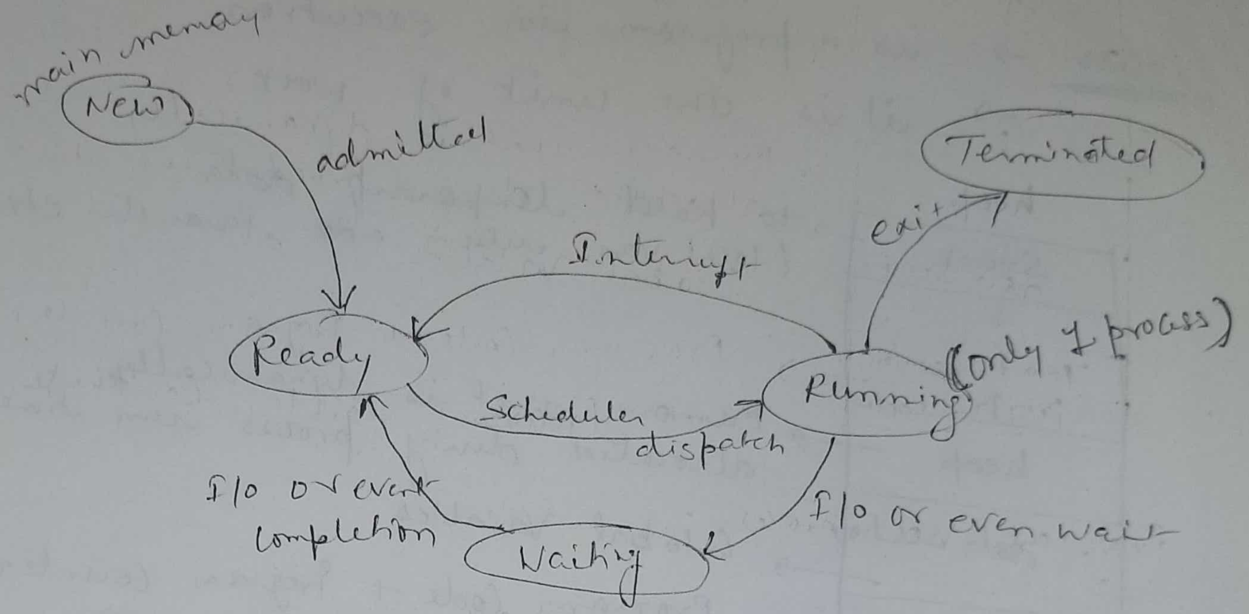
Two Processes may be associated with Same Program are Considered as Separate Processes because their Code may be same but heap, Stack etc. are different.

**Eg** - different copies of mail program.

## Process State -

Current activity of a process is called its **State**.

during execution process changes state.




New - Process is created means it is loaded into main memory.

Running - Instructions are being executed means Process have control of CPU.

Waiting - The process is waiting for some event or I/O device.

Ready - Process is waiting to be assigned to a processor.

Terminated - The process has finished execution.

## Process Control Block (PCB) - (Task Control Block) (2)

every Process represented in OS by its PCB.
it contains informations related to a process -

1. Process state - any from five
2. Program counter - Next insn to be executed.
3. CPU registers - like accumulators, Index registers, stack pointers etc.
4. CPU - scheduling Information - scheduling parameter like Priority
5. Memory - management Info - eg value of base & limit register, Page tables or segment tables.
6. Accounting Info - Process Number (unique), amount of time CPU or other resources used & time limits (allocated time)
7. I/O Status Info - list of I/O devices assigned to process, files open and so on.

| | |
|---|---|
| Process State | |
| Process Number | |
| Program Counter | |
| CPU registers | → Index regi<br>→ Accumulator |
| Memory limits | → base register<br>limit |
| list of open files | |
| Accounting | |

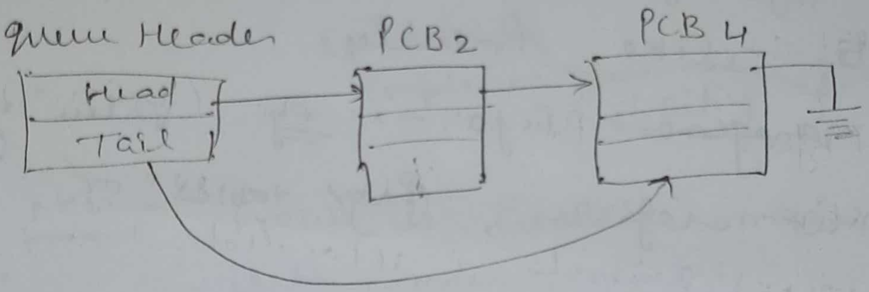
# Process Scheduling -

Process Scheduler selects processes from ready queue to assign CPU.

→ Scheduling queues -

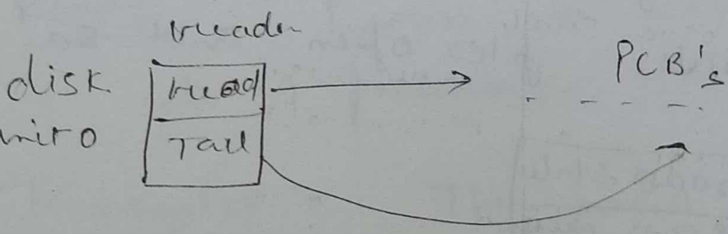
(a) Job queue - Consist all processes in the system (Hard disk).

(b) Ready queue - Contains ~~too~~ processes that are in main memory & waiting for CPU. ready queue is a linklist. its header Points first and final PCB's in the list




Ready queue

(c) Device queue - The list of process waiting for a particular I/o device is Called a device queue.



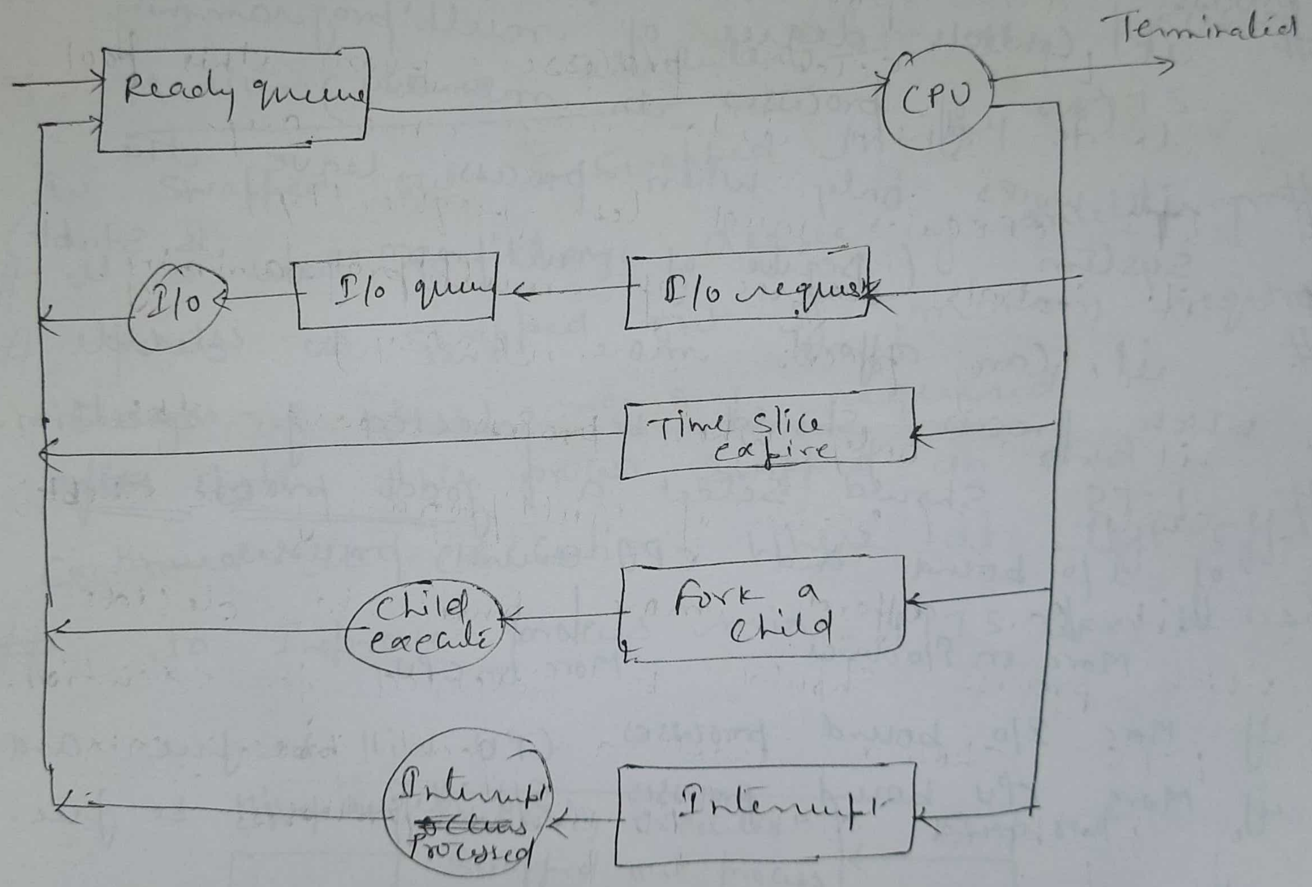

Device queue

queuing diagram -

rectangle - shows queues

Circle - represents resources that serve the queue

arrow - shows flow of processes.




# Initially each process placed in ready queue

# Process continues its cycle untill it terminates after termination its PCB & resources are deallocated & removed from queues.

Schedulers - Selects Job or process from queue

I Long term Scheduler / Job Scheduler -

In a batch system more processes are submitted that can be executed immediately. all these processes are spooled to a disk.

LTS Selects processes from this pool & loads them into main memory.

# LTS executes much less frequently
# it Controls degree of multiprogramming. (No. of processes in memory)
# it works only when process leave the system (Degree of Multiprogramming is Stable)
# it can afford more time to decide which process should be selected for execution.
# LTS should select a good process mix of I/o bound and CPU Bound processes.

I/o bound ↓ More on I/o devices

CPU Bound ↓ More on CPU

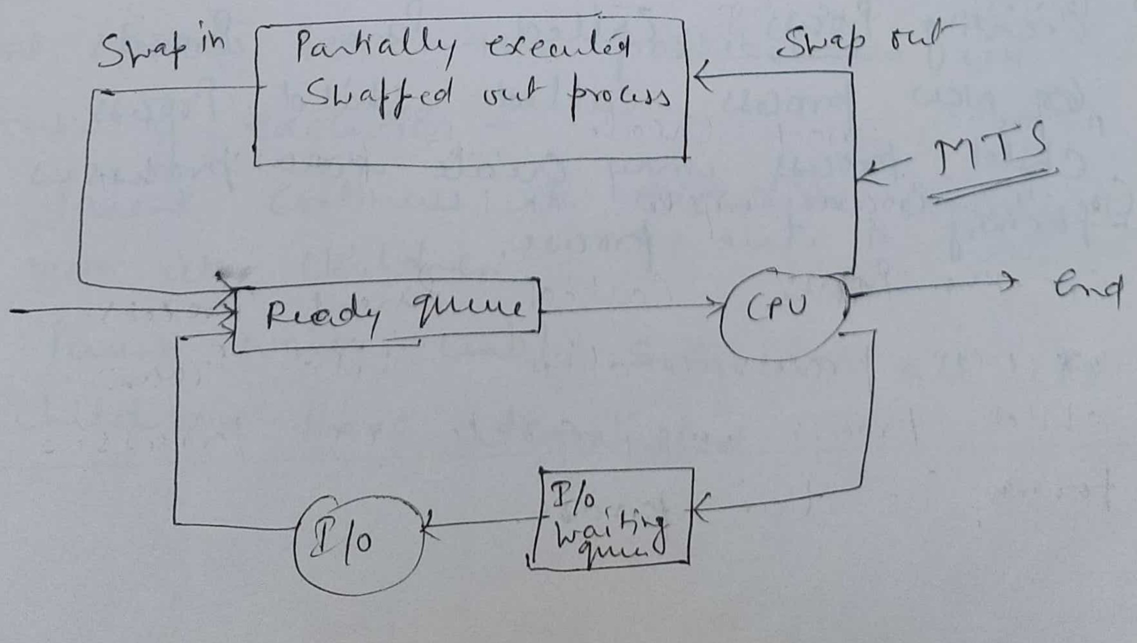
if More I/o bound processes CPU will be free and if More CPU bound processes I/o devices will be free.

2. Short term Scheduler / CPU Scheduler –
it selects from among the processes that are ready to execute and allocates the CPU to one of them.

# STS executes frequently.
# so it should be very fast

3. Medium term Scheduler – the processes are swapped out & swapped in by MTS.

# it is used to change degree of multiprogramming
# process is swapped out if memory requirement changes & free memory required.
~~after~~ Later this process swapped in and continued its execution where it left off.
# to Improve process mix, MTS can be used.



## Context Switch – When an Interrupt occurs or I/O request arises, CPU switches to another process.

In this switching, states of current process are saved and states of new process are restored. This task is known as Context Switch.

# Context switch time is pure overhead.
# its speed vary from M/C to M/C
# Typical speeds are a few milliseconds.
# if extra registes available only Pointer change make context switch.

## Operations on processes –

(a) Process Creation
(b) Process Termination

### (a) Process Creation –

A process may create new process by Create - Process System Call.

Creating Process called Parent Process or New process called Child Process. Child process may create new processes forming a tree process.

each Process have unique Identifier i.e. Pid (an Integer Number)

# each Process need resources, child process may obtain its resources directly from OS or it may be constrained to use only its parent resources.

Process (Parent) may divide its resources among its children.

→ By constrained processes are prevent overloading system by creating many subprocesses

# initialization data may be passed by the parent process to child process.

Eg. child process created to display file contents then file name is passed ~~by~~ as argument by parent process.

# Parent process have two possibilities in terms of execution -

(a) Parent continues its execution concurrently with its children.

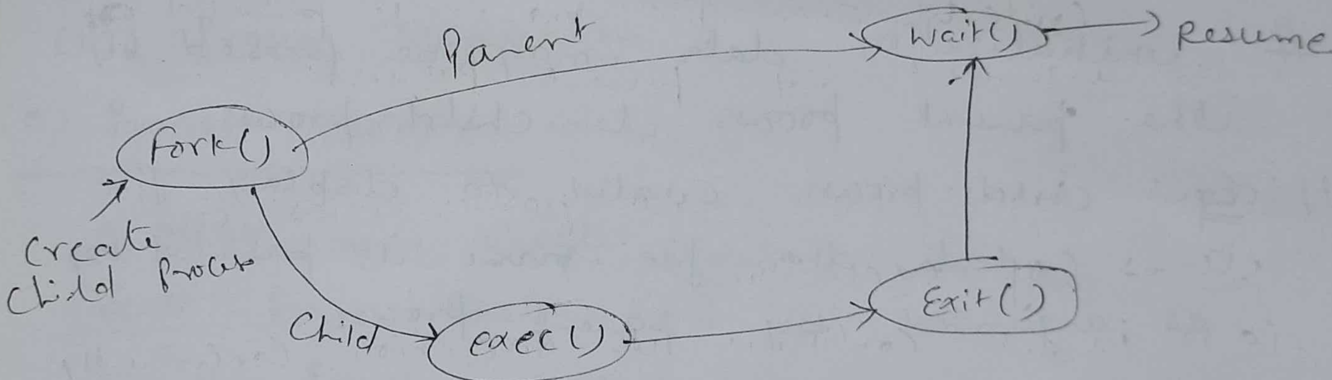
(b) Parent waits until some or all its children have terminated

There are also two possibilities in terms of the address space of the new process –

1. Child process have same program and data as the parent.
2. The child process has a new program loaded into it.

Unix – fork( ) system call is used to create new process.
fork return zero to ~~to~~ child process, return Pid of child process to parent process.

Process Termination –




Process Creation and Termination.

exec() → execute child process
exit() → Child request to OS to delete process(itself)
wait() → Parent waits to terminate its child process. Wait returns terminating child identifier (Pid)

after exit() system call all the resources of child process are deallolated.

# Parent process can terminate its child process. reasons are -

1. Child exceeded its usage of resources
2. Task assigned to child is no longer used
3. The parent is exiting, and OS not allow a child to continue if its parent terminates, this phenomenon known as cascading termination. (initiated by OS)

## Interprocess Communication -

Processes running concurrently are of two types -

(1) Independent - if it can not affect or be affected by the other processes executing in the system, Process is independent. Independent process does not share data with other processes.

(2) Cooperating Processes - vice versa

Process cooperation is required because -

(a) Information sharing - several users may share a file

(b) Computation speed up - task is divided into subtask & run parallely on different processors.

(c) Modularity -

(d) Convinience -

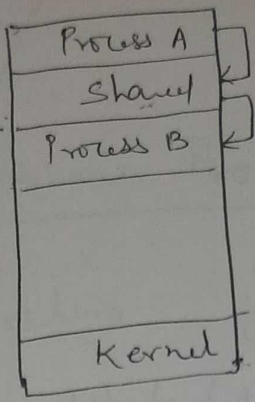
---

Cooperating processes require an IPC mechanism to exchange information -

1. Shared memory - A region of memory is shared between processes

2. Message Passing - message exchanged between processes.

Shared memory system -




# Shared memory allows maximum speed and convienience of communication

# it is faster in a intercomputer ~~comp~~ communication

# system call only required at the shared memory set-up time.

Memory operations

# Once shared region established between processes, after that OS does not have any control means processes are responsible for conflicts (writing at same location)

# Processes agree to remove their restriction of accessing other process address space to share memory.

Example - Producer - Consumer Problem

1. A shared buffer exist
2. Both are synchronized so consumer does not try to consume data that is not produced.

3. Bounded buffer - fixed size
Unbounded buffer - no limit on size.

```
# define BUFFER_SIZE 10
typedef struct {
    - -
} item
item buffer [BUFFER_SIZE];
int in = 0;
int out = 0;
```

(These variables resides in shared memory)

in → Points next free position in buffer

out → points ~~next~~ first full position in buff

( Queue Arrangement)

<u>Produce process</u> -

```
item next produced;
While (true)
  {
  While (((in+1) ÷ BUFFER_SIZE) == out)
        ;
  /* do nothing */
  buffer [in] = next produced;
  in = (in+1) % Buffer_size;
  }
```

---

initially in = 0, out = 0 } $(0+1) \div 10 = 1 == out$ ✗

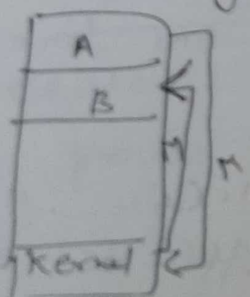
Consumer Process -

```
item  next consumed ;
While (true) {
      While (in == out)
          // do nothing
   next consumed = buffer[out];
   out = (out + 1) % BUFFER_SIZE;
}
```

This Scheme allows at most BUFFER SIZE - 1 items in the buffer at the same time.
(if out = 0 & in = 8 Now next item can be placed at 9th location but condition for producer will not allow it)

Message - Passing System -

# Used in distributed systems.
# Smaller amount of data can be exchanged, Without Conflicts
# Easier to implement, but it is slow because of system calls (Kernel intervention



① Fixed size message
② Variable size message - complex system

Two operations -

Send (Message)
Receive (Message)

Methods to implement logical link between Processes -

① Direct or Indirect Communication

(a) Naming - way to refer each other

direct commn -

Symmetric
Send (P, message) - Send a message to process P
Receive (Q, message) : receive a message from q

Commn link has the following properties -

(a) link is established automatically. Processes need to know each other identities
(b) link is associated with exactly two processes
(c) each pair, their exist one link.

Assymetric
Send (P, message)
receive (id, message) - receive a message from any process.

in ~~naming~~ direct comⁿ if process id changes we have to change all ~~old~~ references.

<u>Indirect communication</u> - Through mailbox

Send (A, message) - Send a message to mailbox A
Receive (A, message) - Receive a message from mailbox A.

<u>Link Properties</u>-

(a) Link is established only if processes share a mail box

(b) One link many processes.

(c) A pair of processes have more than one link (each link with a mail box)

Mailbox owned by -

(a) <u>A process</u> - Process is owner who can receive while other processes can only send message. Process deletes, mail box dele

(b) <u>Operating system</u> - Mail box has an existence of its own.

A process must have mechanism to -

(1)

1. Create a new mail box
2. Send and recieve messages through mailbox
3. delete mail box.

Creating mail box process is owner of mail box, so initially only this process can receive message but ownership and receiving priviledges can be transferred to other processes.

## (2) Synchronization -

- Send () and receive operations may be designed in variety of ways -

(a) Blocking Send - Sending process is blocke until message is received by mail box or receiver

(b) Nonblocking send - Sending process Continues after send ().

(c) Blocking receive - the receiver blocks until a message is available

(d) Non blocking receive - receiver retrieve either Null or valid message.

## (3) Buffering -

(a) Zero Capacity (No Buffering) - queue has length zero. link have no message waiting in it. (Sender blocked)

(b) Bounded capacity
(c) Unbounded Capacity

automatic buffering