|  |  |
| --- | --- |
| What does process include ? |  |
| * Program counter * Stack * Data section |  |

|  |  |
| --- | --- |
| List the process states ? |  |
| * **new**: The process is being created * **running**: Instructions are being executed * **waiting**: The process is waiting for some event to occur * **ready**: The process is waiting to be assigned to a processor * **terminated**: The process has finished execution |  |

|  |  |
| --- | --- |
| What is the information that associated with each process PCB ? |  |
| * **It allow OS to control the process** * **Process state** : new, wait, ready, terminated, run * **Program counter** : when process stop store the instruction address * **CPU registers** : store the process date when process stop * **CPU scheduling information** : priority * **Memory-management information** : limit register (how much) – base register(from what) * **Accounting information** : which user use this process * **I/O status information** : which data the process hold |  |

|  |  |
| --- | --- |
| What is the process data in the memory ? |  |
| * **Stack** : Contains the temporary data such as function program and local variables . * **Heap** : Is memory that is dynamically allocated during the run and it efficient related * **Data** : such as global variable * **Text** : code |  |

|  |  |
| --- | --- |
| Explain the process life cycle ? |  |
| . |  |

|  |  |
| --- | --- |
| What is dispatch responsibility ? |  |
| * Save process state into PCB * Load process state from PCB |  |

|  |  |
| --- | --- |
| Explain the switching between the processes ? |  |
|  |  |

|  |  |
| --- | --- |
| List the process scheduling queues ? |  |
| * **Job queue** : set of all processes in the system * **Ready queue** : set of all processes residing in main memory, ready and waiting to execute * **Device queues** : set of processes waiting for an I/O device   **Processes migrate among the various queues** |  |

|  |  |
| --- | --- |
| What is types of scheduler ? |  |
| * **Long-term scheduler** (or job scheduler) : selects which processes should be brought into the ready queue * is invoked very infrequently (seconds, minutes) ⇒ (may be slow) * It controls the *degree of multiprogramming* * **Short-term scheduler** (or CPU scheduler) : selects which process should be executed next and allocates CPU * is invoked very frequently (milliseconds) ⇒ (must be fast) |  |

|  |  |
| --- | --- |
| What is the types of process ? |  |
| * **I/O-bound process** : spends more time doing I/O than computations, many short CPU bursts * **CPU-bound process** : spends more time doing computations; few very long CPU bursts |  |

|  |  |
| --- | --- |
| What is the context twitch ? |  |
| When CPU switches to another process, the system must save the state of the old(stopped) process and load the saved state for the new process via a **context switch** |  |

|  |  |
| --- | --- |
| What is the context of process ? |  |
| * **Context of a process represented in the PCB** * Process state * SPU register * Memory management info |  |

|  |  |
| --- | --- |
| Is the context switching overhead ? why ? |  |
| **Yes**, because the system does no useful work while switching. And Time dependent on hardware support |  |

|  |  |
| --- | --- |
| Explain the process tree creation ? |  |
| * **Parent** process create **children** processes, which, in turn create other processes, forming a tree of processes |  |

|  |  |
| --- | --- |
| Via what the process identified and managed ? |  |
| via a process identifier (**pid**) |  |

|  |  |
| --- | --- |
| What is the types of Resource sharing between parent and children ? |  |
| * Parent and children share all resources * Children share subset of parent’s resources * Parent and child share no resources |  |

|  |  |
| --- | --- |
| What is the types of Resource sharing between parent and children ? |  |
| * Parent and children execute concurrently (**communication**) * Parent waits until children terminate |  |

|  |  |
| --- | --- |
| Give the UNIX example create the process ? |  |
| * **fork** system call creates new process * **exec** system call used after a **fork** to replace the process’ memory space with a new program |  |

|  |  |
| --- | --- |
| Explain the process creation in diagram ? |  |
|  |  |

|  |  |
| --- | --- |
| explain the process termination ? |  |
| * Process executes last instruction and asks the operating system to delete it (**exit**) * Output data from child to parent (via **wait**) * Process’ resources are deallocated by operating system |  |

|  |  |
| --- | --- |
| why Parent terminate children processes by (**abort**) ? |  |
| * Child has exceeded allocated resources * Task assigned to child is no longer required |  |

|  |  |
| --- | --- |
| What happen for children processes when the parent exiting ? |  |
| * Some operating system do not allow child to continue if its parent terminates , all children terminated - **cascading termination** * Some operating system allow child to continue if its parent terminates |  |

|  |  |
| --- | --- |
| What is the reasons for cooperating processes ? |  |
| * **Information sharing** * **Computation speedup** : in case of multiprocessor * **Modularity** : dividing the system functionality into separate process or threads * **Convenience** : user can work on many program concurrently |  |

|  |  |
| --- | --- |
| What is the IPC and what is the models of it ? |  |
| * Cooperating processes need **interprocess communication** (**IPC**) * Two models of IPC   + Shared memory   + Message passing |  |

|  |  |
| --- | --- |
| What is the types of process in term of sharing ? |  |
| * **Independent** process cannot affect or be affected by the execution of another process * **Cooperating** process can affect or be affected by the execution of another process |  |

|  |  |
| --- | --- |
| Explain cooperating between the process ? |  |
| Paradigm for cooperating processes, producer process produces information that is consumed by a consumer process |  |

|  |  |
| --- | --- |
| What is the types of buffer ? |  |
| * ***unbounded-buffer*** : places no practical limit on the size of the buffer * ***bounded-buffer*** : assumes that there is a fixed buffer size |  |

|  |  |
| --- | --- |
| What is IPC ? |  |
| t is mechanism for processes to communicate and to synchronize their actions. |  |

|  |  |
| --- | --- |
| What is message passing? |  |
| Processes communicate with each other without resorting to shared variables. |  |

|  |  |
| --- | --- |
| What is the operations provided by IPC ? |  |
| * **send**(*message*) – message size fixed or variable * **receive**(*message*) |  |

|  |  |
| --- | --- |
| How the process communicate in message passing ? |  |
| * establish a *communication* *link* between them * exchange messages via send/receive |  |

|  |  |
| --- | --- |
| What is the types of communications in message passing ? |  |
| * **Direct Communication** : Processes must name each other explicitly * **Indirect Communication** : Messages are directed and received from mailboxes |  |

|  |  |
| --- | --- |
| Explain the direct commutation operations ? |  |
| * **Send (P, message)** : send a message to process P * **Receive(Q, message)** : receive a message from process Q |  |

|  |  |
| --- | --- |
| What is the properties of communication link in direct communication ? |  |
| * Links are established automatically * A link is associated with exactly one pair of communicating processes * Between each pair there exists exactly one link * The link may be unidirectional, but is usually bi-directional |  |

|  |  |
| --- | --- |
| Explain the indirect communication ? |  |
| * **Messages are directed and received from mailboxes (also referred to as ports)** * Each mailbox has a unique id * Processes can communicate only if they share a mailbox |  |

|  |  |
| --- | --- |
| What is the properties of communication link in indirect communication ? |  |
| * Link established only if processes share a common mailbox * A link may be associated with many processes * Each pair of processes may share several communication links * Link may be unidirectional or bi-directional |  |

|  |  |
| --- | --- |
| What is the operations in indirect communication ? |  |
| * create a new mailbox * send and receive messages through mailbox * destroy a mailbox   **send**(*A, message*) – send a message to mailbox A  **receive**(*A, message*) – receive a message from mailbox A  where A = mailbox ID |  |

|  |  |
| --- | --- |
| Give three solutions for the following ?   * *P1, P2,* and *P3* share mailbox A * *P1*, sends; *P2* and *P3* receive * Who gets the message? |  |
| * Allow a link to be associated with at most two processes * Allow only one process at a time to execute a receive operation * Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was. |  |