

CPU – Storage Device Interfaces

- Given the general bus hierarchy previously introduced, how does the CPU address an I/O device?
 - I/O devices provide status (read only) and control (write only) registers;
 - Each status/ control word has a unique address.
- 2 basic interfacing techniques,
 - CPU either,
 1. memory maps the I/O to an address space and treats different parts of the address space as read (status) / write (control);
 2. CPU is required to send specific control codes to an I/O device using the corresponding I/O device addresses.
- Interaction with the I/O devices takes one of 3 forms,
 - Polling;
 - Interrupt driven, or;
 - Direct Memory Access.

Polling

- CPU makes request to I/O device (e.g printer / modem)
- CPU repeatedly tests (polls) I/O device status word before next request can be made.
- (dis)advantages,
 - simple – only one bus master;
 - very inefficient – CPU tied up waiting for the I/O device status to change.

Interrupt Driven I/O

- I/O devices can initiate communication (more than one bus master);
- CPU receives a service request in the form of an (external) interrupt.
- CPU (i.e. OS) diagnoses the interrupt source and takes appropriate action.
 - E.g. send more data.
- (dis)advantage,
 - CPU decoupled from I/O device goals, therefore useful use of CPU cycles.

- In real time systems/ transaction processing/ data transfers, 1 000s of interrupts per second result.

Direct Memory Access (DMA)

- Interrupt Driven I/O decouples CPU from initiating communication,
 - But requires CPU to initiate every step of I/O activity.
 - Is this efficient?
- DMA supports goal directed I/O
 - CPU only interrupted when (I/O) task completed.
 - Multiple DMA ‘channels’ used in practice, each with specialist I/O characteristics,
 - Network interface, modem, HD, USB etc..

Reliability, Availability, and Dependability

- Is there a difference between faults, errors, and failures?

S/W example	H/W example
A programming mistake is a fault	An alpha particle hitting a DRAM is considered a fault
As a consequence we have a (latent) error in the code	If memory changes state, a latent error is created
When encountered the error is effective	We have a latent error until the effected word is read, at which point it is effective
Any erroneous data effecting the service is a failure	When the effected word impacts a service, a failure occurs

- Measures of dependability are based on when the system is observed to change from service accomplishment to interruption.
 1. Module reliability
 - Measures continuous service (accomplishment) from predefined initial point in time.
 - Typically expressed by Mean Time to Failure (MTTF) or Rate of Failure (MTTF)⁻¹

- Service interruption is measured by the Mean Time to Repair (MTTR)

2. Module Availability

- Measure of service accomplishment with respect to frequency of service interrupt.

$$\text{Module Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} = \frac{\text{MTTF}}{\text{MTBF}}$$

- Faults themselves naturally have several different sources,
 1. H/W faults – device failure;
 2. Design fault – S/W (norm) or H/W (unusual);
 - Why are S/W faults more usual?
 3. Operational fault – mistakes introduced during maintenance;
 4. Environmental faults – fire, flood, power loss, sabotage...
- Redundancy is the most common approach for improving H/W system reliability.
 - RAID – Redundant Arrays of Inexpensive Disks
- As system complexity increases, most faults are as a result of user behaviour,
 - E.g. maintenance and system ‘upgrades’

Measuring I/O performance

- Measures used to quantify I/O performance attempt to measure a more diverse range of properties than the case of CPU performance.
- Response time (latency) and throughput (bandwidth) are non-linear.
 - From a transaction server model, *Figure 1*,
 1. Throughput is maximized when the queue is never empty;
 2. Response time is minimized when the queue is empty.
 - *Figure 2* summarizes such a relationship.
- Consider two interactive computing environments, one keyboard driven, one graphical.

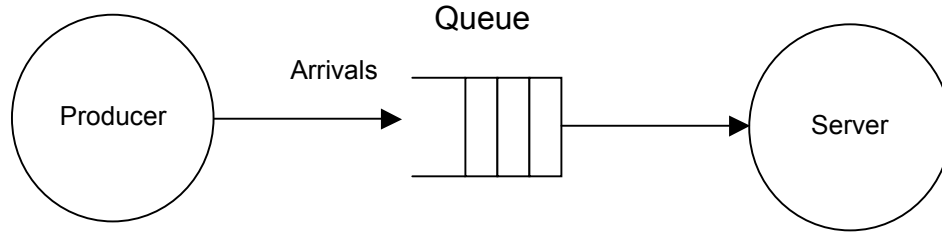


Figure 1 Traditional Producer-Server Model of Response time.

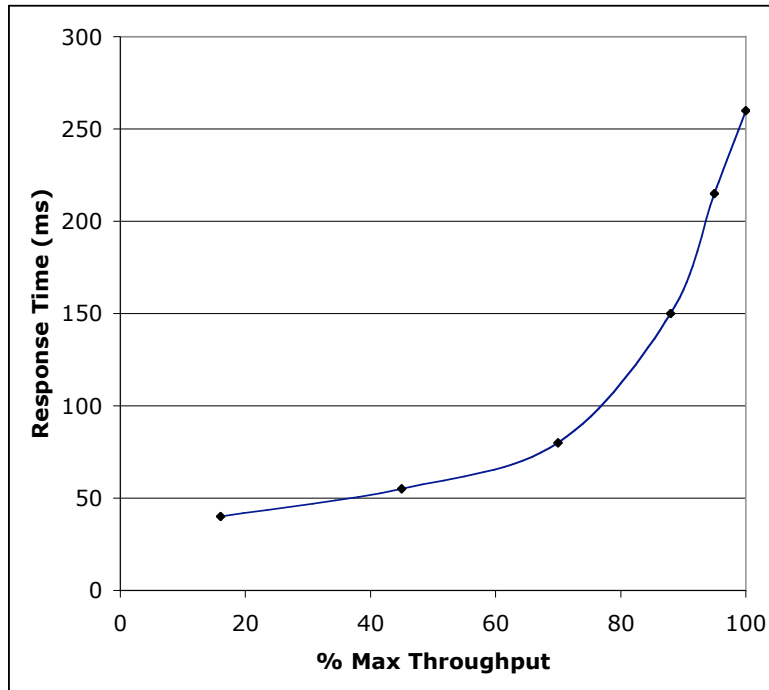


Figure 2 Throughput versus response time for a 'typical' storage system

- Computing interaction or transaction time is divided into three components,
 1. Entry time – time for user to make a request;
 2. System response time – time between request and response;
 3. Think time – time between system response and next request.
- System response time is naturally the shortest duration,
 - Does this minimize the impact of response time?
- *Figure 3* details the contribution of the three properties under different response times for the same task.

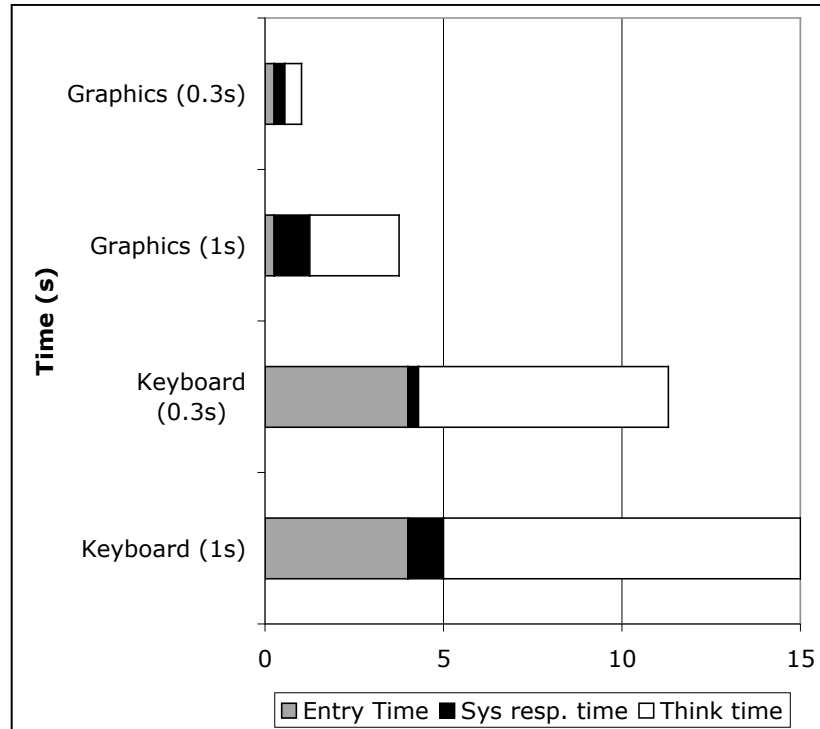


Figure 3. Effect of system response time on user 'thinking' time.

1. Any reduction to response time has more than a linear reduction on total transaction time.
2. Users need less time to think when given a faster response;
3. Possible to attach an economic benefit to response time and throughput;
4. In order to maintain user interest, response times need to be < 1.0 second.

Queue Based Models of IO performance

- Require a new modeling framework for expressing the performance of the IO system.
- Consider the following constraints / features of our IO,
 - Single server model of *Figure 1*;
 - IO request appear as independent events;
 - System is in equilibrium;
 - How realistic are these constraints?
- Given the model of *Figure 1*, basic characteristics of such a system might take the form of,
 - Time(obs) – period over which observations are made;
 - Time(accum) – total time tasks spend in computer system;
 - Num (tasks) – number of tasks completed during the observation period.

- Define,
 - Mean number of tasks in system = Time (accum) ÷ Time (obs)
 - Mean response time = Time (accum) ÷ Num (tasks)
 - Arrival Rate = Num (tasks) ÷ Time (obs)

- Now follows that,

$$\frac{\text{Time(accum)}}{\text{Time(obs)}} = \frac{\text{Time(accum)}}{\text{Num (tasks)}} \times \frac{\text{Num (tasks)}}{\text{Time (obs)}}$$

- Or,

$$\begin{array}{l} \text{Mean \# tasks} \\ \text{in system} \end{array} = \text{Arrival Time} \times \begin{array}{l} \text{Mean Response} \\ \text{Time} \end{array}$$

- Where for equilibrium to hold,
 - Mean # tasks in system < 1
- This is “Little’s Law” from which the system of *Figure 1* has the additional characteristics,
 - Time(server) – Av. Time to service a task.

- Time(queue) – Av. Queue time per task.
- Time(system) – Av. Time per task in the system (or response time)
 - Time(server) + Time(queue)
- Arrival Rate – Av. # of arriving tasks per second.
- Length(server) – Av # tasks in service.
 - Arrival Rate \times Time(server) \rightarrow Server Utilization in the case of a single server system.
- Length(queue) – Av. Queue Length.
- Length(system) - Av. # tasks in the system
 - Length(server) + Length(queue)
- Finally, we need to enforce some sort of queue protocol, where FIFO is the norm.
- FIFO protocol implies that,
 - Time(queue) = Length(queue) \times Time(server) +
Server Utilization \times Av. Time to complete current task
 - Where,
 - *av. Time to complete the current task* is a function of a random event describing the distribution of IO requests.
 - *Server Utilization* is the chance that the server is currently processing a request or not.
- Assuming that such a distribution can be described in terms of the mean and variance, then
$$\text{Av. Time to complete current task} = \frac{\text{Weighted Mean Time of IO Request}}{2} \times (1 + C^2)$$
- Where,
 - C^2 is the square coefficient of variance = Variance \times (Arithmetic Mean)⁻²
 - *Weighted Mean Time of IO Request* is an exponential distribution.
- Notes,
 - If events appear uniformly (not random), then $C = 0$ and the Av. Time to complete current task = $0.5 \times$ Time(server).

- If events appear with an exponential distribution, $C = 1$, and the Av. Time to complete current task = Weighted Mean Time of IO Request
- By recognizing that,
 - Equilibrium also holds for the queue length, thus from Little's Law,
 - $\text{Length}(\text{queue}) = \text{Arrival Rate} \times \text{Time}(\text{queue});$
 - $\text{Server Utilization} = \text{Arrival Rate} \times \text{Time}(\text{server});$
 - Limiting ourselves to the case of exponentially distributed IO requests,
 - We can show that,

$$\text{Time}(\text{queue}) = \text{Time}(\text{server}) \times \frac{\text{Server Utilization}}{(1 - \text{Server Utilization})}$$

- Substituting this expression in for $\text{Length}(\text{queue})$ under equilibrium also implies that,

$$\text{Time}(\text{queue}) = \frac{\text{Server Utilization}^2}{(1 - \text{Server Utilization})}$$

- Substituting the above relation for $\text{Time}(\text{queue})$ and $\text{Server Utilization}$ implies that,
- The above model is actually a specific instance of Markovian queuing model (M/M/1), where generalizations to multi-server models result in a M/M/m Markovian model (where M/ M/ 1 denotes Arrival / Service / # service processes and 'M' is the Markovian process)

Example

A CPU makes 40 disk requests per second. The average service time is 20ms. Assuming an exponential distribution of requests,

1. How utilized is the disk on average?
2. What is the average queue time?
3. What is the average disk response time (i.e. inc. queue and disk service time)?
4. What impact would a new faster disk have if you could decrease the average service time to 10 ms?