Insight into Program execution

सीबीएसई पाठ्यक्रम पर आधारित

कक्षा -11

अध्याय - 15

द्वारा:

संजीव भदौरिया

स्नातकोत्तर शिक्षक (संगणक विज्ञान)

के० वि० बाराबंकी (लखनऊ संभाग)

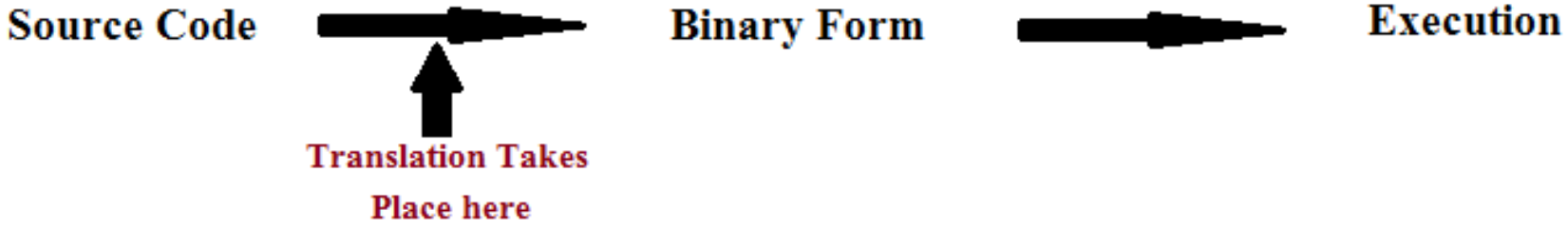
परिचय

- जैसा की हम सभी जानते हैं की कंप्यूटर मशीन और प्रोग्रामर एक दूसरे को नहीं समझते हैं| कहने का तात्पर्य यह है की programmer को कु भाषा आती है उसे कंप्यूटर नहीं जानता और कंप्यूटर जिस भाषा को समझता है वह programmer को नहीं आती|
- Programmer एक कंप्यूटर प्रोग्राम को किसी विशिष्ट भाषा में लिखता है (coding) जैसे C++, JAVA, Python इत्यादि | लेकिन कंप्यूटर इन भाषाओं को सीधे तौर पर नहीं समझ सकता की इनमे क्या निर्देशित है| क्योंकि कंप्यूटर मशीनी भाषा (machine language) समझता है |
- अतः programmer जिस भाषा में निर्देश लिख रहा है, उन निर्देशों के मशीनी भाषा में अनुवादित करना पड़ता है |
- इस अध्याय में हम यही अध्ययन करेंगे कि program लिखने के बाद क्या क्या होता है, program code में कौन कौन से बदलाव किये जाते हैं program क्रियान्वयन अवस्था (Execution state) तक कैसे पहुंचता है तथा program के execution में operating system का क्या योगदान होता है |

Compilation का मूल प्रवाह

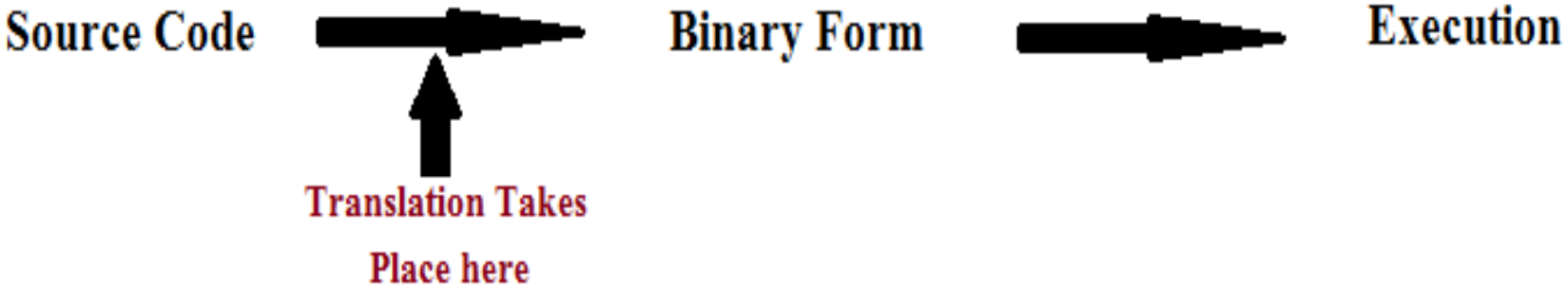
(Basic Flow of Compilation)

- Programmer जब किसी भाषा में program लिखता है तो इसे ***source code*** कहते हैं |
- Source code को प्रोग्रामिंग भाषा के नियमों के अनुसार लिखा जाता है जो की मानव द्वारा आसानी से समझी जा सकती है |
- कंप्यूटर के द्वारा इस source code को समझने के लिए इसे मशीनी भाषा में अनुवादित किया जाता है जिसे ***binary code*** कहते हैं जिसे कंप्यूटर आसानी से समझ सकता है |
- कुल मिला कर हम कह सकते हैं की programmer एक program का source code लिखता है और इसे binary code में अनुवादित किया जाता है ताकि कंप्यूटर इसे execute कर सके |



Translator

- Source Code को binary code में अनुवादित करने के लिए विशेष software की आवश्यकता होती है जो निम्न हैं -
  - Compiler
  - Interpreter
- इन दोनों का कार्य source code को binary code में परिवर्तित करने का है लेकिन दोनों की कार्य करने की शैली अलग अलग है |
- Compiler के काम करने का तरीका interpreter से एक दम भिन्न है |



Compilation

Compiler, source code के लेता है और कुछ सूक्ष्म चरणों में एक low-level-code उत्पन्न करता है | Compilation process के मुख्य चरण निम्न हैं -

1. Processing
2. Compilation
   I. Analysis [front end phase]
   II. Synthesis [back end phase]
3. Assembly
4. Linking
5. Loader

1. Processing

यह अवस्था source code से अतिरिक्त code जैसे comments को हटाती है और उन समस्त ज़रूरी codes को जोडती है जिसकी program में आवश्यकता है ताकि प्रोग्राम execution के लिए पूर्णतया तैयार हो जाये |

2. Compilation

Compilation phase के दो उप-चरण निम्न हैं -

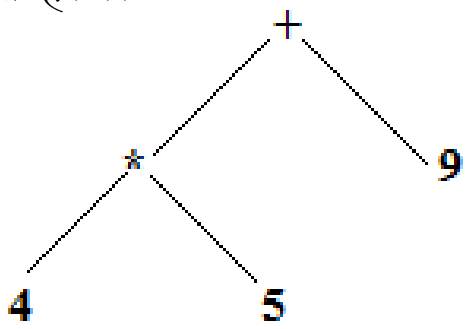
I. Analysis [front end phase]

II. Synthesis [back end phase]

(i) Analysis Phase : analysis phase, source code के समस्त tokens को पता करके उनकी एक संकेत तालिका (symbol table) बना लेता है |

(ii) Synthesis Phase: यह phase, source की पद व्याख्या (parse) करके एक syntax वृक्ष (tree) बनाता है | syntax tree, source code का एक वाक्य-विन्यास ढांचा (syntactic tree) होता है | उदाहरण के लिए -

*4 * 5 + 9 का syntax tree निम्नवत होगा -*



3. Assembly Phase

यह phase कुछ assembly level instruction प्राप्त करके syntax tree को object code में बदल देता है | जोकि मशीन code का एक रूप है |

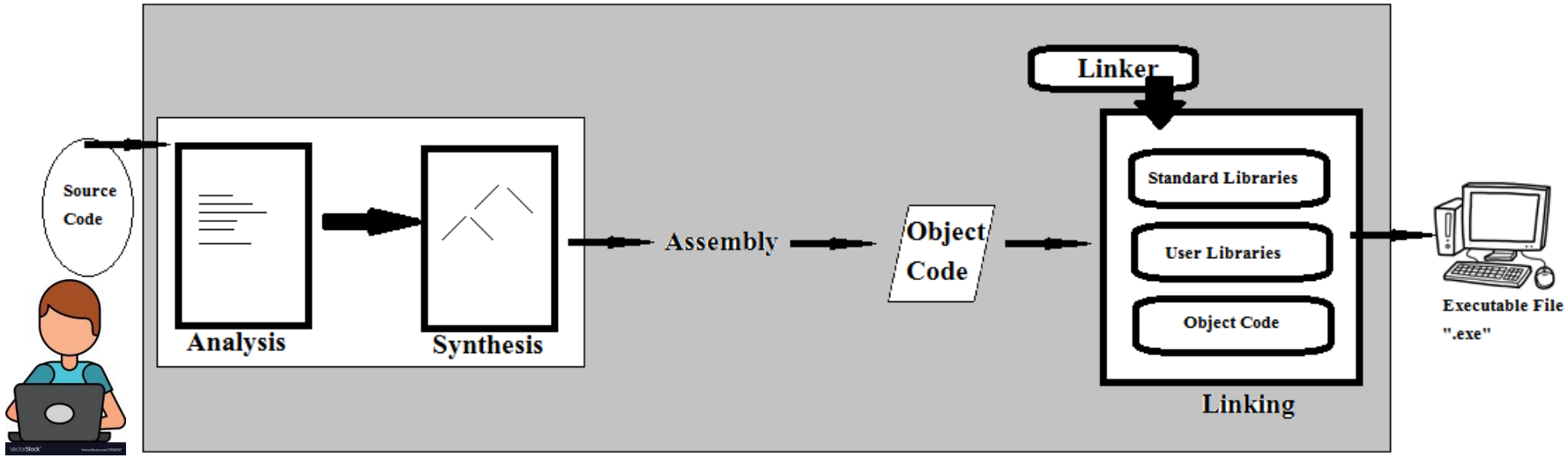
4. Linking

इस phase के द्वारा प्राप्त किया गया code पूरा binary code ही होता है लेकिन कंप्यूटर अभी भी इसको execute नहीं कर सकता क्योंकि महत्वपूर्ण libraries की linking बाकि है जो की अत्यंत आवश्यक कार्य है | यह कार्य ***linker*** के द्वारा पूर्ण किया जाता है |

इस phase में operating system द्वारा समस्त memory references, resolve करके executable (.exe file) बना ली जाती है |

5. Loader

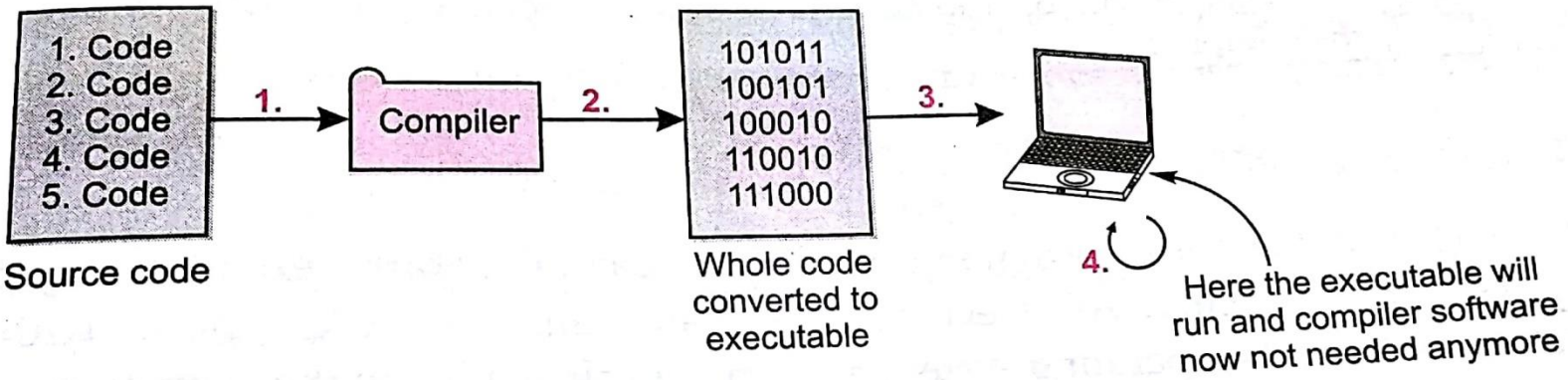
- Loader एक प्रकार का compiler का ही हिस्सा होता है जो मेमोरी में executable file को load करता है |
- अब कंप्यूटर बिना किसी अन्य software की मदद से executable file को run कर सकता है |



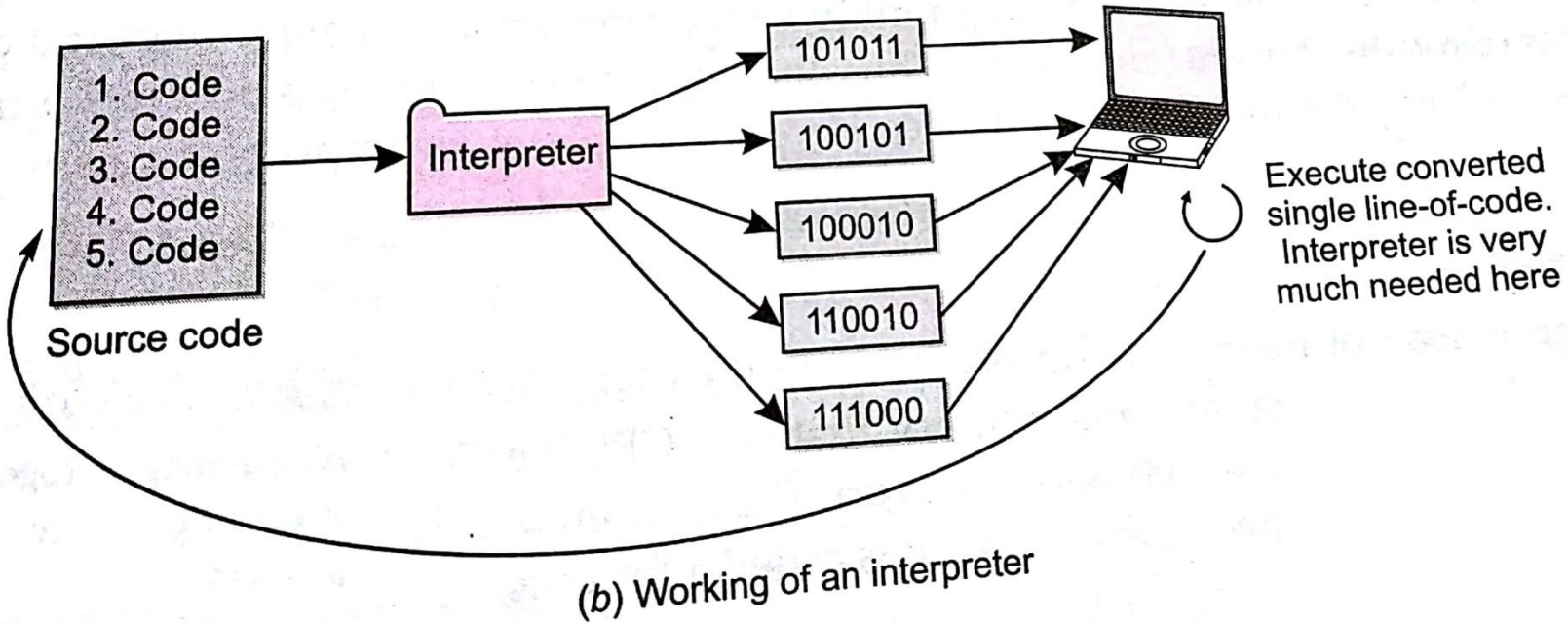
The Interpretation Process

- चूँकि interpreter भी compiler के जैसा ही एक प्रकार का अनुवादक software होता है | इसका कार्य भी source code को मशीन द्वारा समझने योग्य code में बदलना है | compiler के जैसा उद्देश्य होने बावजूद इसका कार्य करने का तरीका compiler से काफी अलग होता है |
- Compiler के उलट यह पूरे program को एक समय में, एक साथ न अनुवादित कर यह एक समय में एक लाइन अथवा एक इकाई को अनुवादित करता है और आवश्यक लाइब्रेरी को सम्मिलित करते हुए run करता है |
- अर्थात दूसरी लाइन के object code में बदलने से पहले पहली लाइन पूर्णतया क्रियान्वित (execute) हो जाती है |
- Interpreter इसी प्रकार प्रत्येक लाइन को ऐसे ही क्रियान्वित करता है |

Compiler बनाम Interpreter



(*a*) Working of a compiler



(*b*) Working of an interpreter

Compiler बनाम Interpreter

Aspects	Compiler	Interpreter
Input	यह पूरे program को एक साथ input लेता है |	यह एक लाइन अथवा एक निर्देश (जैसे loop) को input लेता है |
Output	यह सम्पूर्ण program का एक माध्मिक object code बनाता है |	यह कोई माध्मिक ऑब्जेक्ट code नहीं बनाता है |
Memory	इस क्रिया में अधिक मेमोरी की आवश्यकता होती है |	इस प्रक्रिया में अधिक मेमोरी की आवश्यकता नहीं होती है |
Errors	यह सम्पूर्ण program की सारी त्रुटियाँ एक साथ लाइन number के साथ दिखाता है |	यह एक एक लाइन में त्रुटि दिखाता हुआ आगे बढ़ता है |
Always Required	जैसे ही एक program compile होगया उसके बाद compiler की आवश्यकता नहीं होती है |	इसमें interpreter की सदैव आवश्यकता होती है | जितनी बार program run होगा उतनी बार interpreter की अवश्यकता होगी
Workload	Compiler की आवश्यकता बार बार नहीं होने से workload कम होता है |	इसमें interpreter की बार बार आवश्यकता होने से workload बढ़ जाता है |

Program के run होने में Operating System की भूमिका

- जैसा की हम हानते हैं की जब कोई program, executable(.exe) के रूप में आ जाता है तो उसको कंप्यूटर स्वतन्त्र रूप से execute कर सकता है |
- और यह executable file कंप्यूटर में operating system की सहायता से run होती है |
- आज कल multi-user और multi-processing के समय में किसी प्रोग्राम को run करने में operating system का महत्व बहुत बढ़ जाता है साथ ही जटिलता भी बढ़ जाती है |
- Operating system किसी भी user और hardware के बीच में मध्यस्थता का कार्य करता है |
- तथा OS किसी program के run होने के लिए एक प्लेटफार्म भी प्रदान करता है |

OS एक श्रोत व्यवस्थापक (Resource Manager)के रूप में

- जब program, running अवस्था में न हो तो इसे program कहते हैं | तथा जब कोई program, active state अर्थात running अवस्था में हो तो उसे ***process*** कहते हैं | कहीं कहीं इसे ***job*** भी कहते हैं |
- जब एक से अधिक program, run हो रहे होते हैं तो OS सारे श्रोतों (resources) को manage करके कंप्यूटर की कार्यक्षमता को बढाता है | जिसे मापने के लिए throughput शब्द का प्रयोग किया जाता है |

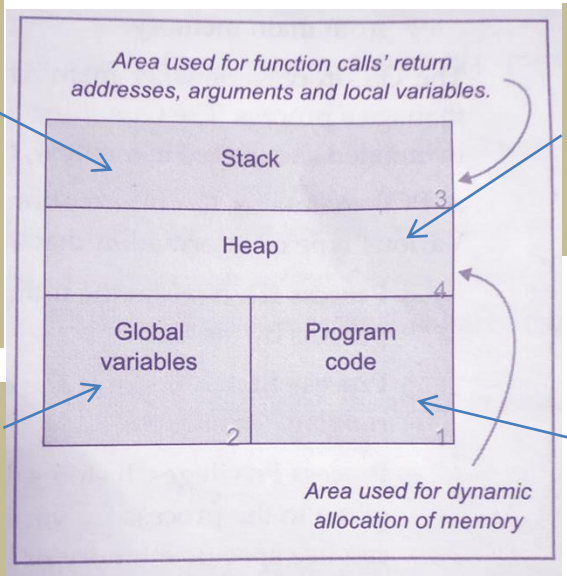
$$\text{Throughput} = \frac{\text{कुल समाप्त job की संख्या}}{\text{लिया गया समय}}$$

- Multiple Processing के लिए OS निम्न फंक्शन प्रदान करता है
  - Process Management
  - Process Scheduling
  - Memory Management
  - I/O Management

Process Management

- जब एक program को execute होना होता है तो यह OS के पास पहुचता है |
- OS का ***process scheduler*** इसे मेमोरी में load करता है जिससे यह प्रोसेस बन जाती है | जो कि अपने निर्देशों के अनुसार execute होती है |
- किसी प्रोसेस के लिए मेमोरी में निम्न खंड बनते हैं |

यहाँ CPU के state के साथ साथ समस्त लोकल variable, Parameters, फंक्शन कॉल के return addresses इत्यादि रखे जाते हैं | इसे stack के नाम से भी जाना जाता है इसमें program के एक्षेकुतिओन के समय की बहुत सी चीज़ें राखी जाती हैं |



यहाँ से dynamic memory allocate करने के लिए मेमोरी प्राप्त की जाती है |

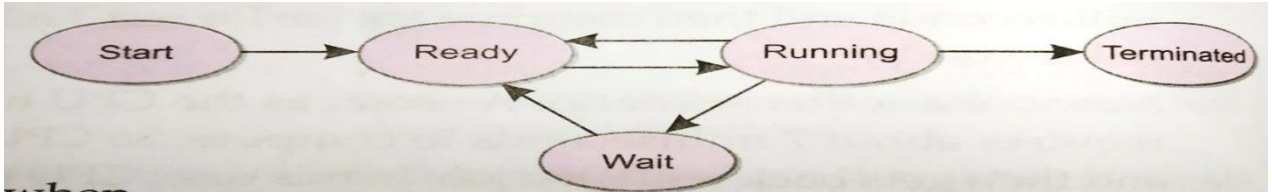
यहाँ program के समस्त global variables को रखा जाता है | ये variable program के अंत तक मेमोरी में रहते हैं |

यह क्षेत्र program का compile किया हुआ code रखता है|

Process Management

- एक program जब execute होने के लिए तैयार होता है तो यह मेमोरी में load **(start)** कर दिया जाता है तब इसकी स्थिति active प्रोसेस की हो जाती है | और यह active process विभिन्न स्थितियों से गुज़रती है |

- **Start State:** यह स्थिति तब आती है जब program को मेमोरी में load करके सबसे पहले start कर दिया जाता है |
- **Ready State:** यह स्थिति तब आती है जब प्रोसेस execute होने के लिए तैयार होती है और अपनी बारी के लिए CPU का इंतज़ार कर रही होती है |
- **Running State:** यह स्थिति तब आती है जब process को CPU time मिल जाता है और CPU इस प्रोसेस को execute करने लग जाता है |
- **Waiting State:** Execution के दौरान यदि प्रोसेस को किसी श्रोत की आवश्यकता पड़ती है तो waiting state आ जाती है जैसे user के द्वारा input देने के लिए इंतज़ार करना |
- **Terminated अथवा Exit:** जैसे ही किसी प्रोसेस के समस्त निर्देश CPU द्वारा पूरे कर दिये जाते हैं तथा प्रोसेस का execution पूर्ण हो जाता है तो इस प्रोसेस को OS के द्वारा terminate कर दिया जाता है | और terminate की गयी प्रोसेस को मेमोरी से remove कर दिया जाता है |



Process Management

- OS प्रोसेस मैनेजर एक प्रोसेस को manage करने के लिए एक data structure बनाता है जिसे Process Control Block (PCB) कहते हैं |
- OS प्रत्येक प्रोसेस के लिए PCB बनाता है |
- जैसे ही कोई प्रोसेस मेमोरी में load होती है तो इसको एक ID (इसे Process ID अर्थात PID कहते हैं)प्रदान करके PCB बना दी जाती है |
- PCB का काम प्रोसेस के बारे में महत्वपूर्ण सूचना रखना है ताकि प्रोसेस का track रखा जा सके | PCB निम्न सूचनाएं रखता है |

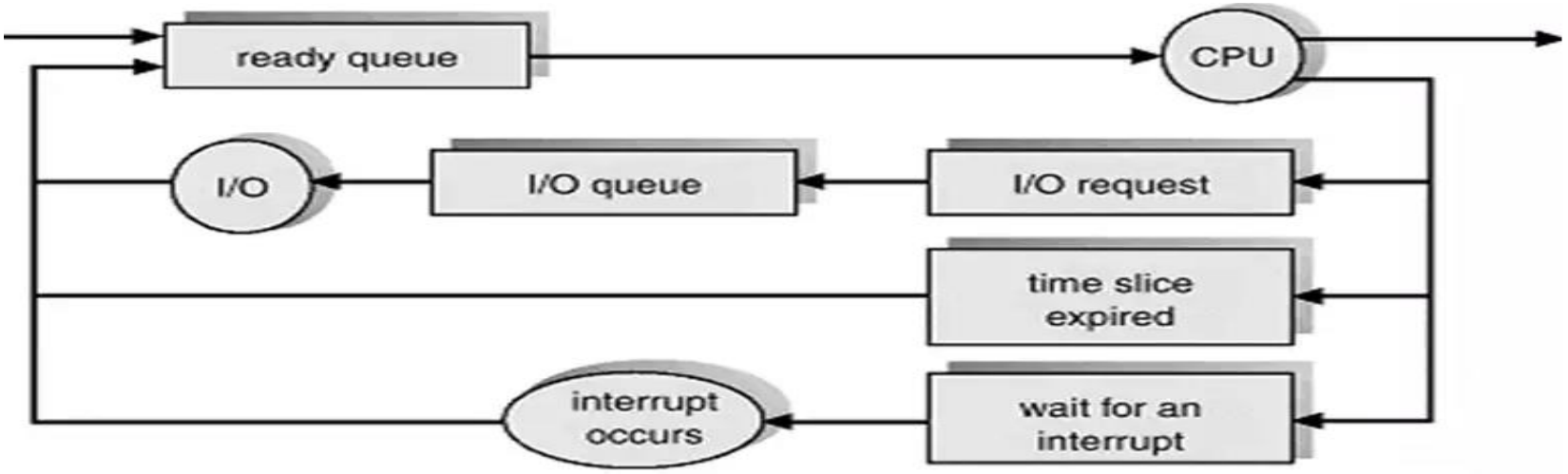
Process ID	OS द्वारा प्रदत्त Process ID को store करता है	
State	State -> ready, running, waiting ….इत्यादि	
Pointer	Pointer , sub-processes के address को रखता है	
Priority	कम या ज्यादा Priority वाले processes का डाटा	
Program Counter	Program Counter अगले निर्देश का address संग्रहीत करता है	
CPU Registers	CPU Registers, execution के लिए सूचना रखता है	
Etc . . .	Etc . . .	

OS के द्वारा Process Scheduling

- यह तो हम सभी जानते हैं की कंप्यूटर की मेमोरी में execution के लिए एक साथ कई program load किये जा सकते हैं | ऐसे case में in processes को manage करना अत्यंत महत्वपूर्ण हो जाता है | जिसके लिए निम्न बातें पता करना आवश्यक है -
  - किस प्रोसेस को CPU time मिलेगा ?
  - किस प्रोसेस को CPU से remove किया जाना है ?
  - किस प्रोसेस को I/O के लिए shift करना है ?
  - यदि कोई हस्तक्षेप(interrupt) होता है तो क्या करना है ?
- यहाँ interrupt CPU के लिए एक संकेत (signal) होता है, कि किसी उच्च वरीयता वाली प्रोसेस को तुरंत CPU चाहिए |
- यह सारा कार्य OS के process scheduler का होता है |
- Process scheduling, process manager की एक गतिविधि होती है जिसके अंतर्गत प्रोसेस manager CPU से running process को remove करता है और एक कार्यनीति (stratagy) के तहत अन्य process का चुनाव करता है |

OS के द्वारा Process Scheduling . . . .

- प्रोसेस के चुनाव की कुछ नीतियां निम्न हैं –
  - पहले आओ पहले पाओ (FCFS)
  - राउंड रोबिन शेड्यूलिंग (Round Robin Scheduling)
  - सबसे छोटी पहले (Shortest Job Next) इत्यादि |
- इनके अलावा भी बहुत सी नीतियाँ होती हैं
- प्रोसेस scheduler कुछ Process Scheduling Queues को भी बनाता है
- **Job Queue** : system की समस्त प्रोसेस की लाइन
- **Ready queue**: ready state के साथ प्रोसेस की लाइन
- **Device Queues :** जैसे I/O की लाइन



Memory Management

- चूँकि किसी भी प्रोसेस होने वाली चीज़ को मेमोरी में load किया जाता है अतः OS का Memory Management वाला हिस्सा भी बहुत महत्वपूर्ण है|
- मेमोरी मैनेजर निम्न कार्यों को पूरा करता है –
  - प्रत्येक memory लोकेशन का पता संग्रहीत रखता है |
  - यह पता रखता है की प्रोसेस को कितनी मेमोरी दी(allocate) जानी है और कितनी दी जा चुकी है |
  - यह निर्धारित करता है कि किस प्रोसेस को कब मेमोरी दी जायेगी |
  - प्रदत्त मेमोरी में डाटा की सुरक्षा का दायित्व |
- कुछ मेमोरी मैनेजमेंट की तकनीकियाँ निम्न हैं –
- Single Memory Allocation
- Partitioned memory allocation
- Paged Memory Management
- Segmented Memory Management

I/O Management

- जैसे ही किसी प्रोसेस को input या output की आवश्यकता होती है तो OS का यह हिस्सा ज़िम्मेदारी लेता है |
- I/O Manager, CPU के पास एक request भेजता है और CPU के लिए एक interupt उत्पन्न होता है |
- CPU running प्रोसेस को I/O के लिए भेज देता है तथा तब तक CPU अन्य प्रोसेस को पूरा करता है जैसे ही पहली प्रोसेस का I/O पूरा हो जाता है, CPU पुनः इस प्रोसेस को पूरा करने में लग जाता है |

PARALLEL COMPUTING

- जब कंप्यूटर ईजाद हुए थे तब उनमे एक ही processor होता था जो एक समय में एक job ही पूरा कर पता था| एक से अधिक job को पूरा करने के लिए एक के बाद एक पूरा करता था | तकनीकी में सुधर और विकास के साथ साथ processor खुद में भी अधिक सक्षम हुए तथा कंप्यूटर भी एक से अधिक processor के साथ आने लगे |
- जो की एक साथ एक ही समय में एक से अधिक job को पूरा करने लगे |
- एक ही समय में एक से धिक् processor या कंप्यूटर के द्वारा किये जाने वाले job के पूरा करने के प्रयोग को ***PARALLEL COMPUTING*** कहा जाता है|
- एक parallel computing system में -
  - एक प्रॉब्लम को कई आत्मनिर्भर हिस्सों में बांटा जाता है
  - प्रत्येक हिस्से में इसके खुद के निर्देश होते हैं जिन्हें execute किया जाता है|
  - प्रत्येक हिस्से के निर्देश अलग अलग processor के द्वारा पूरा किया जाता है |
- ऐसा करने के लिए या तो एक से अधिक processor वाला कंप्यूटर प्रयोग में लाया जाता है या नेटवर्क में एक से अधिक कंप्यूटर काम में लाये जाते हैं |
- ऐसा करने से **समय की बचत**, बड़ी से बड़ी और **जटिल समस्या आसानी से हल** होती है, **hardware का पूरा प्रयोग** होता है तथा **दूर के श्रोतों** को भी प्रयोग में लाया जा सकता है |

CLOUD COMPUTING

- **क्लाउड कम्प्यूटिंग** (Cloud Computing) या **मेघ संगणना** वास्तव में इंटरनेट-आधारित प्रक्रिया और कंप्यूटर ऐप्लीकेशन का इस्तेमाल है।
- गूगल एप्स क्लाउड कंप्यूटिंग का एक उदाहरण है जो बिजनेस ऐप्लीकेशन ऑनलाइन मुहैया कराता है और वेब ब्राउजर का इस्तेमाल कर इस तक पहुंचा जा सकता है।
- यह दो प्रकार का होता है -
- Public Cloud जैसे Google Drive, iCloud इत्यादि
- Private Cloud जैसे किसी व्यापारिक संगठन का cloud जैसे KVS का शालादर्पण

धन्यवाद

और अधिक पाठ्य-सामग्री हेतु निम्न लिंक पर क्लिक करें -

www.pythontrends.wordpress.com