|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OS** คือSystemที่รวม ComputerHardware กับ Userเข้าด้วยกัน ทำหน้าที่จัดการทรัพยากรทั้งหมดที่มีเพื่อให้ User ใช้งานได้สะดวกและมีประสิทธิภาพ การทำงานของOSประกอบด้วย UserInterface DeviseManagement FileManagement MemoryManagement ProcessManagement Communication Security  **UserInterface** คือการติดต่อกับผู้ใช้ 1.Batch(การประมวรผลแบบกลุ่ม) 2.CommandLine(MS),Terminal(Unix) 3.GraphicalUser(ปัจจุบัน)  **SystemCallกับAPI** SCคือส่วนประสานระหว่างโปรแกรมประยุกต์กับKernel APIมีหน้าที่ในการสั่งการOSโดยตรง SCเป็นสับเซ็ตของAPI | **Process** คือพื้นที่ในการRunProgramที่OSสร้างขึ้น  **-การสร้าง Process** fork() (Unix), CreateProcess() (MS)   |  |  | | --- | --- | | 1.#include <stdio.h>  2.#include <unistd.h>  3.int main(int argc, char \*argv[])  4.{  5. int pid;  6. /\* create another process \*/  7. pid = fork();  8. if (pid < 0) { /\* error occurred \*/  9. fprintf(stderr, "Fork Failed");  10. exit(-1); | 11. }  12. else if (pid == 0) { /\* child process \*/  13. execlp("./child","child",NULL);  14. }  15. else { /\* parent process waits for \*/  16. /\* the child to complete \*/  17. wait(NULL);  18. printf("Child completed");  19. exit(0);  20. }} |   ต่างกันที่ forkจะสร้างลูกออกมาจากแม่ CreateProcess จะสร้างขึ้นมาเฉยๆ  **-Process Termination** จะจบการทำงานได้จาก exit(), TimeQuota(หมดเวลา),  ละเมิดหน่ายความจำProcessอื่นๆ(Traps()), I/Oมีความผิดพลาด,กระบวนการเป็นโมฆะ  ,Processแม่Terminateแล้ว(บางOSจะเป็นแบบนี้) | **Process Scheduling** คือการเลือกหรือการเปลี่ยนProcessที่จะให้CPUทำการรัน  ขยายจากการDispatchของบทที่2 ในบทนี่มีสองmode  (**Non-preemptive** ทำprocessให้เสร็จทีละตัวโดยไม่มีเปลี่ยน จนกว่าจะเสร็จ) (**Preemptive** OSจะเข้ามาในการจัดการเวลาที่Processแต่ละตัวจะได้ใช้CPU)  Processor utilization = อัตราการใช้งานCPU  Throughput = ปริมาณงานที่ทำเสร็จในช่วงเวลาหนึ่ง  Turnaround time = ระยะเวลาที่ทำ Job นั้นเสร็จ  Waiting time = ระยะเวลาที่ process รอใช้ CPU  Response time = ระยะเวลาที่ส่ง job ไปแล้ว result เริ่มกลับมา  Variance = ความแปรปรวน |
| **Kernel** แปลว่าใจ,แกนกลาง มีหน้าที่ในการเชื่อมUserLevelกับHardware และอีกหน้าที่หนึ่งคือSystemProtection |S|H|->|U|K||H| เป้าหมายคือ1.ป้องกันไม่ให้Processทั้งหลายใช้ทรัพยากรร่วมกัน 2.ป้องกันKernelจากUserProgramที่เกเร 3.หลีกเลี่ยงการใช้MemoryของProgramอื่น SystemProtectionมี4methods  1.dual-Mode แบ่งการทำงานของCPUเป็นUserMode(มีข้อจำกัด)กับKernelModeโดยมีModeBitคอยบอกสถานะ 2.MemoryProtectionไม่ให้ProcessไปAccessเข้าMemoryAddressที่Processอื่น BaseRegisterกับLimitRegister(ความกว้าง)  3./4. I/OกับCPU Protection I/Oทุกตัวเป็นสมบัติของOSซึ่งต้องรับในKernelModeเท่านั้น | Inside a Process มีส่วนประกอบหลักๆคือ Code, Data, Stack, ProgramCounter, RegisterRepository, TemporarySpace (ทั้งหมดนี้อยู่ในdisk)  ProcessState & LifeCircle  Admit  dispatch  running  ready  Event Occurs  Terminated  Time-Out  Activate  Suspended ready  Suspended block  Suspended  Wait  Exit  blocked  new | **First-Come, First-Served** **(FCFS)** Processที่มาก่อนจะได้ใช้CPUก่อน  ข้อเสียคือถ้างานแรกใช้CPUนานWaiting-timeจะสูง  **Round-Robin(RR)** มี time quantum คอยแบ่งการทำงานของแต่ละ process  โดยปกติจะมี time quantumอยู่ประมาณ 10-100 milliseconds ข้อเสียคือในการสลับไปทำ Process อื่นจะเสียเวลานิดหน่อย ถ้าแบ่งเยอะเกิน ก็เสียเวลามาก  quantum = 5    **Waiting-time** = [(P1=0+10) + (P2=5+9+5+2) + (P3=10+9+5)] / 3 = 18.33 |
| **EvolutionOfOS** MainFrame->PC+ClientServer->Distibuted,Wireless จบ  SimpleBatchProcessing-->Multi-User,Multi-Programing--> Multi-User ,Time-Sharing->SingleUser,PC-->DistibutedSystem-->Web-besed-->Wireless,clound  1.SerialProcessing(1940-50) (No OS) (ManualOperation)  2.BatchProcessing (fed job one-by-one) (Job Control Language)  3.Multiprogramming(multitasking) จะหว่างที่PSรอI/Oก็ให้CPUทำอันอื่นก่อน  4.TimeSharing เครื่องเดียวใช้หลายคนแต่เหมือนใช้คนเดียว  **OS(80-90)** (PC) (Multiprocessors processorหลายตัว) (Distributed Systems หลายเครื่องต่อกันด้วยNetwork) (Peer-to-Peerไม่มีServergเครื่องไหนมีไฟล์ก็เป็นไป)  (ComputerClusterตู้CPUหลายตู้แต่จอเดียว) (VirtualMachinesที่ลงUbuntu)  **OS(2000+)** (WirelessDistributedSystem ColundหรือServerRoom)  (Hand-HeldComputers ก็คือโทรศัพท์) (Computer Tablets)  (Container Virtualization – e.g. Docker) | **Essential OS Data Structures**  **Process** จะใช้ computer resources หลายแบบ ได้แก่ Processor, Memory, Files  ,I/O devices | **Shortest-Job-First (SJF)** เลือกตาม process ที่มี burst-time น้อยสุด  **แบบ Preemptive นะ**  Average waiting time  = (9 + 1 + 0 +2) / 4 = 3  **Priority Scheduling** จะเลือกตามระดับความสำคัญของ Process  Unix(เลขน้อย = สำคัญมาก), Windows(เลขมาก = สำคัญมาก) บอกว่าOSไหนด้วยละ  ข้อเสียคือ ถ้าคิวที่ความสำคัญมากๆ มี process เข้ามาไม่หยุด อันหลังๆจะไม่ได้ทำ  **Multilevel Queue** เหมือนกับ **Priority Scheduling** แต่ถ้า Priorityไหนไม่ได้ทำนานๆ ก็จะถูกเลื่อนขั้น โดยlevelที่สำคัญๆจะมีTimeQuantumสั้นๆและยาวขึ้นตามชั้น  **\*\*\* ตอนคำนวณ waiting-time อย่าลืมเอาค่า arrival time มาลบด้วยนะ** |
| StructorOfAnOS (4types)  -Simple โปรแกรมประยุกต์ใช้Hardwareได้โดยตรง  -Layered ใช้พท.น้อย แต่ละชิ้นต่อกันแค่สองทาง,นานกว่าจะเข้าถึงชั้นLayerที่ต้องการ  -Microkernel แปรงมาจากSimpleแต่ย้ายอันที่เข้าถึงยากๆออกมาที่UserSpace  -Modular สามารถติดต่อกันได้เลยโดยไม่ต้องผ่านตัวกลาง  -Mixed  **คำถามท้ายบท**  -สามเหตุการณ์ที่ทำให้OSเปลี่ยนเป็นKernelMod(Interupt, fault occurs, switches)  cat แสดงผลข้อมูลภายใน file ในรูปแบบ text   |  |  | | --- | --- | | chmod เปลี่ยนสิทธิ์ในกำรเข้าถึง file | cksum นับจำนวน bytes ของ file | | df แสดงข้อมูลพื้นที่ disk ทั้งหมด | clear ล้างหน้ำจอ screen | | echo แสดงผลบนหน้าจอ screen | Cmp เปรียบเทียบfilesในระดับbytes | |  | **Fair Share Scheduling** คำนึงถึงผู้ใช้(หลายคน) ผู้ใช้แต่ละคนจะได้ใช้CPUในเวลาที่เท่าๆกัน เช่น ผู้ใช้2คน ได้คนละ50% แม้ว่าผู้ใช้คนแรกจะมี process มากกว่าก็ตาม  **Real-Time Scheduling** ใช้ในงานที่ต้องการความแม่นยำสูง แบ่งได้เป็น 2 กลุ่ม  -Soft real-time งานที่สำคัญจะได้รับค่า priority และทำงานจนกว่าจะลุล่วง  -Hard real-time ต้องยืนยันว่างานท่สำคัณจะต้องเสร็จในเวลาต้องควบคุม delay ให้ได้  คำถามท้ายบท  เวลาครบวงงาน = |
| |  |  | | --- | --- | | dir แสดงข้อมูล directory | dirname แสดงชื่อdirectory ของ file | | du ดูข้อมูลรำยละเอียดขนาด file | Findใช้ในการค้นหาfileหรือdirectory | | foldเรียงความยาวอักษรแต่ละบรรทัด | free แสดงข้อมูลกำรใช้งาน memory | | hostnameแสดงชื่อของเครื่องserver | kill ส่ง Signal หรือยกเลิกprocess | | ln สร้าง link เชื่อมโยงกันระหว่าง file | ls แสดงข้อมูลภายใน directory | | mkdir สร้าง directory | mv ย้ายตำแหน่ง file หรือ directory | | nl แสดงเลขที่บรรทัดของข้อมูลใน file | passwd เปลี่ยน password | | ps แสดง process ต่าง ๆ | pwd แสดง directory ที่อยู่ปัจจุบัน | | rm ลบ file หรือ directory | rmdir ลบ directory | | shutdown ปิดกำรทำงานของระบบ | whoami แสดงชื่อ user ที่ใช้ login | | sort จัดเรียงข้อมูลfileทีละบรรทัด | touch เปลี่ยนแปลงfileTimeStamps | | wc นับจำนวนคำและบรรทัดจำก file | ***watch***MonitorProcessที่ทำงานน้อย | | **Context Switching**  …🡪Save process context in PCB🡪Change states information🡪Select next running process🡪Rearrange PCB links🡪Update to-berunning process🡪Set PC to run the next process🡪…  **Direct vs. Indirect Message Passing**  Direct คนส่งกับคนรับต้องถึงพร้อมกัน ต้องรอกัน มีPIDกันโดยตรง ไม่ปลอดภัย  Indirect ทีส่วนกลางให้ เหมือน mail box ข้อเสียคือเปลืองเนื้อที่  **Synchronization Mode** เป็นเหมือนกับ mode ในการรับส่งข้อมูล  Synchronous ต้องรอ, Asynchronous ไม่ต้องรอ  **Thread** ไม่แน่ใจรู้แค่ Thread จะอยู่ใน Process อีกที เหมือนเป็นสิ่งที่ process จะต้องทำ ยิ่งเยอะยิ่งดี,สำคัญ |  |
| Field public java.awt.Image carvequest #Data XREF : init+182 Line arrow StraightW …  Field public java.awt.Image inst #Data XREF : init+196 Line arrow StraightW …  # =====================================================  #Segment type : Pure code  Method public void init()  Max\_stack 6  Max\_locals 4  {  aload\_0 # var001\_0  aload\_0 # var001\_0  invokevirtual java.awt.Dimension java.awt.Component.size()  getfield int java.awt.Dimension.width  putfield int width  aload\_0 # var001\_0  aload\_0 # var001\_0  invokevirtual java.awt.Dimension java.awt.Component.size()  getfield int java.awt.Dimension.height  putfield int height  aload\_0 # var001\_0  ldc1 “GRIDWIDTH”  invokevirtual java.lang.String java.applet.getParameter(java.lang.String)  astore\_1 # var001\_1 | คำถามท้ายบท  Forkจะได้ stack, heap, memory ร่วมกับแม่  การสร้าง Thread จะต้องใช้อะไรบ้าง |  |