PROCESOS Y ADMINISTRACION DEL PROCESADOR

- INTRODUCCION Y DEFINICIONES SOBRE PROCESOS
- ESTADOS DE PROCESO
- PROCESAMIENTO DE INTERRUPCIONES
- EL NUCLEO DEL S. O.
- COMUNICACIÓN ENTRE PROCESOS
- CONCURRENCIA DE EJECUCION Y PLANIFICACION DE PROCESOS
- NIVELES DE PLANIFICACION DEL PROCESADOR
- OBJETIVOS DE LA PLANIFICACION
- CRITERIOS DE PLANIFICACION
- PLANIFICACION APROPIATIVA VERSUS NO APROPIATIVA
- TEMPORIZADOR DE INTERVALOS O RELOJ DE INTERRUPCIONES

PROCESOS Y ADMINISTRACION DEL PROCESADOR

- PRIORIDADES
- TIPOS DE PLANIFICACION
- MULTIPROCESAMIENTO
- ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR
- GRADOS DE ACOPLAMIENTO EN MULTIPROCESAMIENTO
- S. O. DE MULTIPROCESADORES
- RENDIMIENTO DEL SISTEMA DE MULTIPROCESAMIENTO
- RECUPERACION DE ERRORES
- MULTIPROCESAMIENTO SIMETRICO
- TENDENCIAS DE LOS MULTIPROCESADORES

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

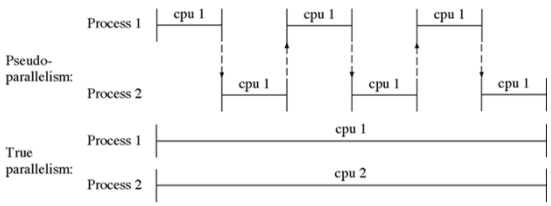
INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

- EL **CONCEPTO CENTRAL** DE CUALQUIER S. O. ES EL DE "**PROCESO**": UNA **ABSTRACCION DE UN PROGRAMA EN EJECUCION** TAMBIEN LLAMADA "**TAREA**".
- NO HAY UN ACUERDO UNIVERSAL SOBRE UNA DEFINICION DE PROCESO, PERO SI **ALGUNAS DEFINICIONES ACEPTADAS**:
  - UN **PROGRAMA** QUE SE ESTA **EJECUTANDO**.
  - UNA **ACTIVIDAD ASINCRONICA**.
  - EL "**EMPLAZAMIENTO DEL CONTROL**" DE UN **PROCEDIMIENTO** QUE ESTA SIENDO **EJECUTADO**.
  - AQUELLO QUE SE MANIFIESTA POR LA EXISTENCIA EN EL S. O. DE UN "**BLOQUE DE CONTROL DE PROCESO**".
  - AQUELLA **ENTIDAD** A LA CUAL SON **ASIGNADOS LOS PROCESADORES.**
  - LA UNIDAD "**DESPACHABLE**".

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

- EN SISTEMAS DE **MULTIPROGRAMACION** LA **CPU ALTERNA** DE PROGRAMA EN PROGRAMA, EN UN ESQUEMA DE **SEUDOPARALELISMO**:
  - LA CPU EJECUTA EN CIERTO **INSTANTE UN SOLO PROGRAMA**, INTERCAMBIANDO MUY RAPIDAMENTE ENTRE UNO Y OTRO.
- EL **PARALELISMO REAL DE HARDWARE** SE DA:
  - EN EJECUCION DE **INSTRUCCIONES DE PROGRAMA** CON **MAS DE UN PROCESADOR** DE INSTRUCCIONES EN USO **SIMULTANEAMENTE**.
  - CON LA **SUPERPOSICION** DE EJECUCION DE **INSTRUCCIONES DE PROGRAMA** CON LA EJECUCION DE UNA O MAS **OPERACIONES DE E / S.**
- EL OBJETIVO ES **AUMENTAR EL PARALELISMO** EN LA EJECUCION.

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

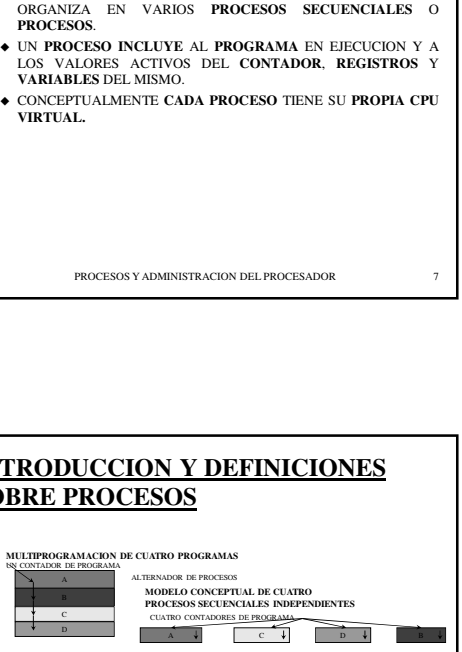


PSEUDO-PARALELISMO Y VERDADERO PARALELISMO

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

- **EL MODELO DE PROCESOS:**
  - TODO EL SOFTWARE EJECUTABLE, INCLUSIVE EL S. O., SE ORGANIZA EN VARIOS **PROCESOS SECUENCIALES** O **PROCESOS**.
  - UN **PROCESO INCLUYE** AL **PROGRAMA** EN EJECUCION Y A LOS VALORES ACTIVOS DEL **CONTADOR**, **REGISTROS** Y **VARIABLES** DEL MISMO.
  - CONCEPTUALMENTE **CADA PROCESO** TIENE SU **PROPIA CPU VIRTUAL.**

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS



INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

- **SI LA CPU ALTERNA** ENTRE LOS PROCESOS LA **VELOCIDAD** A LA QUE **EJECUTA UN PROCESO NO SERA UNIFORME**:
  - LOS **PROCESOS NO** DEBEN PROGRAMARSE CON **HIPOTESIS IMPLICITAS** ACERCA DEL **TIEMPO**.
  - NORMALMENTE **LA MAYORIA DE LOS PROCESOS NO SON AFECTADOS POR LA MULTIPROGRAMACION** SUBYACENTE DE LA CPU O LAS VELOCIDADES RELATIVAS DE PROCESOS DISTINTOS.
- UN **PROCESO** ES UNA **ACTIVIDAD DE UN CIERTO TIPO**, QUE TIENE UN **PROGRAMA**, **ENTRADA**, **SALIDA** Y **ESTADO**.
- UN **SOLO PROCESADOR** PUEDE SER **COMPARTIDO ENTRE VARIOS PROCESOS** CON CIERTO "**ALGORITMO DE PLANIFICACION**":
  - DETERMINA CUANDO DETENER EL TRABAJO EN UN PROCESO Y DAR SERVICIO A OTRO DISTINTO.

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

- **JERARQUIAS DE PROCESOS**:
  - LOS S. O. DEBEN DISPONER DE UNA FORMA DE **CREAR Y DESTRUIR PROCESOS** CUANDO SE REQUIERA DURANTE LA OPERACION.
  - LOS PROCESOS PUEDEN GENERAR **PROCESOS HIJOS** MEDIANTE LLAMADAS AL S. O., PUDIENDO DARSE EJECUCION EN PARALELO.

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

- **ESTADOS DEL PROCESO:**
  - CADA **PROCESO** ES UNA **ENTIDAD INDEPENDIENTE** PERO FRECUENTEMENTE DEBE **INTERACTUAR** CON OTROS PROCESOS.
  - LOS PROCESOS PUEDEN **BLOQUEARSE** EN SU EJECUCION PORQUE:
    - DESDE EL PUNTO DE VISTA LOGICO **NO PUEDE CONTINUAR** (ESPERA DATOS QUE AUN NO ESTAN DISPONIBLES).
    - EL S. O. ASIGNO LA **CPU A OTRO PROCESO**.

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

- LOS ESTADOS QUE PUEDE TENER UN PROCESO SON:
  - **EN EJECUCION**: UTILIZA LA CPU EN EL INSTANTE DADO.
  - **LISTO**: EJECUTABLE, SE DETIENE EN FORMA TEMPORAL PARA QUE SE EJECUTE OTRO PROCESO.
  - **BLOQUEADO**: NO SE PUEDE EJECUTAR DEBIDO A LA OCURRENCIA DE ALGUN EVENTO EXTERNO.
- SON POSIBLES CUATRO **TRANSICIONES** ENTRE ESTOS ESTADOS.

INTRODUCCION Y DEFINICIONES SOBRE PROCESOS

UN PROCESO PUEDE ESTAR EN EJECUCION, BLOQUEADO O LISTO

TANSICIONES ENTRE LOS ESTADOS

1-EL PROCESO SE BLOQUEA EN ESPERA DE DATOS
2-EL PLANIFICADOR ELIGE OTRO PROCESO
3-EL PLANIFICADOR ELIGE ESTE PROCESO
4-LOS DATOS ESTAN DISPONIBLES

1-BLOQUEO
2-TIEMPO EXCEDIDO
3-DESPACHO
4-DESPERTAR

ESTADOS DE PROCESOS

ESTADOS DE PROCESOS

- DURANTE SU EXISTENCIA UN **PROCESO PASA** POR UNA SERIE DE **ESTADOS** DISCRETOS:
  - VARIAS **CIRCUNSTANCIAS** PUEDEN HACER QUE UN **PROCESO CAMBIE DE ESTADO**.
  - SE PUEDE ESTABLECER UNA "**LISTA DE LISTOS**" PARA LOS PROCESOS LISTOS Y UNA "**LISTA DE BLOQUEADOS**" PARA LOS BLOQUEADOS.
  - LA "**LISTA DE LISTOS**" SE MANTIENE EN **ORDEN PRIORITARIO**.
  - LA "**LISTA DE BLOQUEADOS**" ESTA **DESORDENADA**:
    - LOS PROCESOS SE DESBLOQUEAN EN EL ORDEN EN QUE TIENEN LUGAR LOS EVENTOS QUE ESTAN ESPERANDO.

ESTADOS DE PROCESOS

- AL **ADMITIRSE UN TRABAJO** EN EL SISTEMA SE **CREA UN PROCESO** EQUIVALENTE Y ES INSERTADO EN LA ULTIMA PARTE DE LA **LISTA DE LISTOS**.
- LA **ASIGNACION DE LA CPU** AL PRIMER PROCESO DE LA "**LISTA DE LISTOS**" SE DENOMINA "**DESPACHO**":
  - ES EJECUTADO POR UNA ENTIDAD DEL S. O. LLAMADA "**DESPACHADOR**".
- EL **BLOQUEO** ES LA UNICA **TRANSICION DE ESTADO** INICIADA POR EL **PROPIO PROCESO** DEL USUARIO:
  - LAS OTRAS TRANSICIONES SON INICIADAS POR ENTIDADES AJENAS AL PROCESO.

ESTADOS DE PROCESOS

- LA **MANIFESTACION DE UN PROCESO** EN UN S. O. ES UN "**BLOQUE DE CONTROL DE PROCESO**" (**PCB**) CON INFORMACION QUE INCLUYE:
  - **ESTADO** ACTUAL DEL PROCESO.
  - **IDENTIFICACION** UNICA DEL PROCESO.
  - **PRIORIDAD** DEL PROCESO.
  - **APUNTADORES** PARA LOCALIZAR LA **MEMORIA** DEL PROCESO.
  - **APUNTADORES** PARA ASIGNAR **RECURSOS**.
  - **AREA** PARA PRESERVAR **REGISTROS**.
- CUANDO EL **S. O. CAMBIA LA ATENCION DE LA CPU** ENTRE LOS PROCESOS, UTILIZA LAS **AREAS DE PRESERVACION DEL PCB** PARA MANTENER LA **INFORMACION** QUE NECESITA PARA **REINICIAR** EL PROCESO CUANDO CONSIGA DE NUEVO LA CPU.

ESTADOS DE PROCESOS

- LOS **SISTEMAS QUE ADMINISTRAN LOS PROCESOS** DEBEN PODER:
  - CREAR, DESTRUIR, SUSPENDER, REANUDAR, CAMBIAR LA PRIORIDAD, BLOQUEAR, DESPERTAR Y DESPACHAR UN PROCESO.
- LA **CREACION DE UN PROCESO** SIGNIFICA:
  - DAR **NOMBRE** AL PROCESO.
  - INSERTAR UN PROCESO EN LA **LISTA** DEL SISTEMA DE **PROCESOS CONOCIDOS**.
  - DETERMINAR LA **PRIORIDAD INICIAL** DEL PROCESO.
  - CREAR EL **BLOQUE DE CONTROL** DEL PROCESO.
  - ASIGNAR LOS **RECURSOS INICIALES** DEL PROCESO.

ESTADOS DE PROCESOS

- UN PROCESO PUEDE "**CREAR UN NUEVO PROCESO**":
  - EL PROCESO CREADOR SE DENOMINA "**PROCESO PADRE**".
  - EL PROCESO CREADO SE DENOMINA "**PROCESO HIJO**".
  - SE OBTIENE UNA "**ESTRUCTURA JERARQUICA DE PROCESOS**".
- LA **DESTRUCCION DE UN PROCESO** IMPLICA:
  - **BORRARLO** DEL SISTEMA.
  - **DEVOLVER** SUS RECURSOS AL SISTEMA.
  - **PURGARLO** DE TODAS LAS LISTAS O TABLAS DEL SISTEMA.
  - **BORRAR** SU BLOQUE DE CONTROL DE PROCESOS.

ESTADOS DE PROCESOS

- UN **PROCESO SUSPENDIDO** NO PUEDE PROSEGUIR HASTA QUE OTRO PROCESO LO REANUDE.
- **REANUDAR** (REACTIVAR) UN PROCESO IMPLICA **REINICIARLO** EN EL PUNTO DONDE FUE SUSPENDIDO.
- LA **DESTRUCCION DE UN PROCESO** PUEDE O NO SIGNIFICAR LA DESTRUCCION DE LOS PROCESOS HIJOS, SEGUN EL S. O.
- GENERALMENTE SE DENOMINA **TABLA DE PROCESOS** AL CONJUNTO DE **INFORMACION DE CONTROL** SOBRE LOS DISTINTOS **PROCESOS**.

ESTADOS DE PROCESOS

Gestión de procesos	**Gestión de Memoria**	**Gestión de Ficheros**
Registros	Puntero al segmento de código	Directorio raíz
Contador de programa	Puntero al segmento de datos	Directorio de trabajo
Palabra de estado del programa	Puntero al segmento de pila	Descriptores de ficheros
Puntero de pila		UID
Estado del proceso		GID
Prioridad		
Parámetros de planificación		
Identificar de proceso		
Proceso padre		
Grupo del proceso		
Señales		
Tiempo de inicio del programa		
Tiempo de CPU consumido		
Tiempo de CPU de los hijos		
Tiempo para la siguiente alarma		

EJ. DE CONTENIDO DE UNA TABLA DE PROCESOS

PROCESAMIENTO DE INTERRUPCIONES

PROCESAMIENTO DE INTERRUPCIONES

- UNA "**INTERRUPCION**" ES UN **EVENTO QUE ALTERA LA SECUENCIA** EN QUE EL **PROCESADOR** EJECUTA LAS INSTRUCCIONES:
  - ES UN HECHO **GENERADO POR EL HARDWARE** DEL COMPUTADOR.
- **CUANDO OCURRE** UNA INTERRUPCION **EL S. O.:**
  - **OBTIENE EL CONTROL**.
  - **SALVA EL ESTADO** DEL PROCESO INTERRUMPIDO:
    - GENERALMENTE EN SU **BLOQUE DE CONTROL** DE PROCESOS.
  - **ANALIZA** LA INTERRUPCION.
  - **TRANSFIERE EL CONTROL** A LA RUTINA APROPIADA PARA LA MANIPULACION DE LA INTERRUPCION.

PROCESAMIENTO DE INTERRUPCIONES

- UNA **INTERRUPCION** PUEDE SER **INICIADA** POR:
  - UN **PROCESO** EN ESTADO DE EJECUCION.
  - UN **EVENTO** QUE PUEDE O NO ESTAR RELACIONADO CON UN PROCESO EN EJECUCION.

PROCESAMIENTO DE INTERRUPCIONES

- GENERALMENTE LAS **INTERRUPCIONES** SE PUEDEN CLASIFICAR POR **TIPOS** SEGUN EL SIGUIENTE DETALLE:
  - **"SVC (LLAMADA AL SUPERVISOR)":**
    - ES UNA PETICION GENERADA POR EL USUARIO PARA UN **SERVICIO PARTICULAR DEL SISTEMA**, POR EJ.:
      - REALIZACION DE E / S, OBTENCION DE MAS MEMORIA.
  - **"E / S":**
    - SON INICIADAS POR EL **HARDWARE DE E / S**, INDICANDO A LA CPU QUE HA **CAMBIADO EL ESTADO DE UN CANAL O DISPOSITIVO**, POR EJ.:
      - FINALIZACION DE E / S, OCURRENCIA DE UN ERROR.
  - **"EXTERNAS":**
    - SON CAUSADAS POR **DISTINTOS EVENTOS**, POR EJ.:
      - **EXPIRACION DE UN CUANTO** EN UN RELOJ DE INTERRUPCION.
      - RECEPCION DE UNA **SEÑAL DE OTRO PROCESADOR** EN UN SISTEMA MULTIPROCESADOR.

PROCESAMIENTO DE INTERRUPCIONES

  - **"DE REINICIO":**
    - OCURREN AL PRESIONAR LA "**TECLA DE REINICIO**" O CUANDO LLEGA UNA **INSTRUCCION DE REINICIO DE OTRO PROCESADOR** EN UN SISTEMA MULTIPROCESADOR.
  - **"DE VERIFICACION DE PROGRAMA":**
    - SON CAUSADAS POR **ERRORES** PRODUCIDOS DURANTE LA **EJECUCION** DE PROCESOS, POR EJ.:
      - UN INTENTO DE **DIVIDIR POR CERO**.
      - UN INTENTO DE UN PROCESO DE USUARIO DE EJECUTAR UNA **INSTRUCCION PRIVILEGIADA**.
      - UN INTENTO DE EJECUTAR UN **CODIGO DE OPERACION INVALIDO**.
  - **"DE VERIFICACION DE MAQUINA":**
    - SON OCASIONADAS POR UN **MAL FUNCIONAMIENTO** DEL HARDWARE.

PROCESAMIENTO DE INTERRUPCIONES

- EL **S. O.** INCLUYE RUTINAS LLAMADAS "**MANIPULADORES DE INTERRUPCIONES (IH)**" PARA PROCESAR CADA TIPO DIFERENTE DE INTERRUPCION.
- **CUANDO SE PRODUCE** UNA INTERRUPCION EL S. O.:
  - **SALVA EL ESTADO** DEL PROCESO INTERRUMPIDO.
  - DIRIGE EL **CONTROL AL MANIPULADOR** DE INTERRUPCIONES ADECUADO.
  - SE APLICA LA TECNICA DE "**CAMBIO DE CONTEXTO**".
- LOS S. O. INSTRUMENTAN **INFORMACION DE CONTROL** QUE PUEDE APARECER COMO LAS "**PALABRAS DE ESTADO DE PROGRAMA (PSW)**":
  - CONTROLAN EL **ORDEN DE EJECUCION** DE LAS INSTRUCCIONES.
  - CONTIENEN INFORMACION SOBRE EL **ESTADO DEL PROCESO**.
- EXISTEN **TRES TIPOS DE PSW**:
  - ACTUAL, NUEVA Y VIEJA.

PROCESAMIENTO DE INTERRUPCIONES

- LA "**PSW ACTUAL**":
  - ALMACENA LA **DIRECCION DE LA PROXIMA INSTRUCCION** QUE SERA EJECUTADA.
  - INDICA LOS TIPOS DE INSTRUCCIONES **ACTUALMENTE** "**HABILITADAS**" E "**INHABILITADAS**".
- EN UN **SISTEMA UNIPROCESADOR** EXISTE:
  - **SOLO UNA PSW ACTUAL**.
  - **SEIS PSW NUEVAS** (UNA PARA CADA TIPO DE INTERRUPCION).
  - **SEIS PSW VIEJAS** (UNA PARA CADA TIPO DE ITERRUPCION).
- LA **PSW NUEVA** PARA UN TIPO DE **INTERRUPCION DADO** CONTIENE LA **DIRECCION** EN EL HARDWARE DONDE RESIDE EL **MANIPULADOR DE INTERRUPCIONES** PARA ESTE TIPO ESPECIFICO.

PROCESAMIENTO DE INTERRUPCIONES

- CUANDO OCURRE UNA **INTERRUPCION** PARA LA CUAL EL **PROCESADOR NO ESTA INHABILITADO**:
  - EL HARDWARE **CAMBIA LAS PSW** EN LOS CASOS SIGUIENTES:
    - AL **ALMACENAR LA PSW ACTUAL EN LA PSW VIEJA**, PARA ESTE TIPO DE INTERRUPCION.
    - AL **ALMACENAR LA PSW NUEVA EN LA PSW ACTUAL**, PARA ESTE TIPO DE INTERRUPCION.

PROCESAMIENTO DE INTERRUPCIONES

- LUEGO DE ESTE "**INTERCAMBIO DE PSW**":
  - LA **PSW ACTUAL** CONTIENE LA DIRECCION DEL **MANIPULADOR DE INTERRUPCION** ADECUADO.
  - EL MANIPULADOR DE INTERRUPCIONES **PROCESA LA INTERRUPCION.**
  - **LUEGO** DE PROCESAR LA INTERRUPCION LA **CPU ES ENVIADA AL:**
    - **PROCESO QUE ESTABA EN EJECUCION** EN EL MOMENTO DE LA INTERRUPCION.
    - **PROCESO DE LISTO** DE MAS ALTA **PRIORIDAD**.
  - **DEPENDE** DE SI EL PROCESO DE INTERRUPCION ES:
    - "**APROPIATIVO**": OBTIENE LA CPU SOLO SI NO HAY PROCESOS DE LISTOS.
    - "**NO APROPIATIVO**": OBTIENE DE NUEVO LA CPU.

EL NUCLEO DEL SISTEMA OPERATIVO

EL NUCLEO DEL SISTEMA OPERATIVO

- EL "**NUCLEO**" DEL S. O. CONTROLA TODAS LAS OPERACIONES QUE IMPLICAN **PROCESOS**.
- REPRESENTA SOLO UNA **PEQUEÑA PORCION** DEL CODIGO DE TODO EL S. O. PERO ES DE **AMPLIO USO**.
- **GENERALMENTE** PERMANECE EN EL **ALMACENAMIENTO PRIMARIO.**
- **EL PROCESO DE INTERRUPCIONES** SE INCLUYE EN EL **NUCLEO**:
  - DEBE SER **RAPIDO** (ESPECIALMENTE EN SISTEMAS MULTIUSUARIO) PARA:
    - **OPTIMIZAR** EL USO DE LOS RECURSOS DEL SISTEMA.
    - PROVEER **TIEMPOS DE RESPUESTA** ACEPTABLES A LOS USUARIOS INTERACTIVOS.
- EL **NUCLEO INHABILITA** LAS INTERRUPCIONES **MIENTRAS RESPONDE** A UNA INTERRUPCION:
  - LAS INTERRUPCIONES **SON HABILITADAS DE NUEVO** DESPUES DE COMPLETAR EL PROCESO DE UNA INTERRUPCION.

EL NUCLEO DEL SISTEMA OPERATIVO

- EL **NUCLEO** DEL S. O. GENERALMENTE REALIZA LAS SIGUIENTES **FUNCIONES**:
  - MANIPULACION DE **INTERRUPCIONES**.
  - **CREACION** Y **DESTRUCCION** DE **PROCESOS**.
  - CAMBIO DE **ESTADOS** DE PROCESOS.
  - **DESPACHO**.
  - **SUSPENSION** Y **REANUDACION** DE PROCESOS.
  - **SINCRONIZACION** DE PROCESOS.
  - **COMUNICACION** ENTRE PROCESOS.
  - **MANIPULACION** DE BLOQUES DE CONTROL DE PROCESO.
  - SOPORTE DE LAS **ACTIVIDADES** DE E / S.

EL NUCLEO DEL SISTEMA OPERATIVO

- SOPORTE DE LA ASIGNACION Y DESASIGNACION DE **ALMACENAMIENTO**.
- SOPORTE DEL **SISTEMA DE ARCHIVOS**.
- SOPORTE DE UN MECANISMO DE **LLAMADA / REGRESO AL PROCEDIMIENTO.**
- SOPORTE DE CIERTAS FUNCIONES CONTABLES (**ESTADISTICAS**) DEL SISTEMA.

COMUNICACIÓN ENTRE PROCESOS

COMUNICACIÓN ENTRE PROCESOS

- **INTRODUCCIÓN**
- LOS PROCESOS NECESITAN **COMUNICARSE** CON OTROS PROCESOS.
- HAY TRES **CUESTIONES**:
  - CÓMO UN PROCESO PUEDE **PASAR INFORMACIÓN** A OTRO.
  - HACER QUE DOS O MÁS PROCESOS **NO SE INTERPONGAN** ENTRE SÍ CUANDO COMPITEN POR EL MISMO RECURSO.
  - OBTENER LA **SECUENCIA APROPIADA** CUANDO HAY DEPENDENCIAS PRESENTES:
    - SI EL PROCESO *A* **PRODUCE** DATOS Y EL PROCESO *B* LOS **IMPRIME**, *B* TIENE QUE ESPERAR HASTA QUE *A* HAYA PRODUCIDO ALGUNOS DATOS ANTES DE EMPEZAR A IMPRIMIR.
- LOS MISMOS PROBLEMAS Y SOLUCIONES SE APLICAN A LOS **HILOS**.

COMUNICACIÓN ENTRE PROCESOS

- **CONDICIONES DE CARRERA**
- OCURREN CUANDO **DOS O MÁS PROCESOS** ESTÁN LEYENDO O ESCRIBIENDO ALGUNOS **DATOS COMPARTIDOS** Y EL **RESULTADO FINAL DEPENDE** DE QUIÉN SE EJECUTA Y EXACTAMENTE CUÁNDO LO HACE.
- OCURRE CUANDO UN **PROCESO** EMPIEZA A **UTILIZAR** UNA DE LAS **VARIABLES COMPARTIDAS ANTES** DE QUE OTRO PROCESO TERMINE DE UTILIZARLA.

COMUNICACIÓN ENTRE PROCESOS

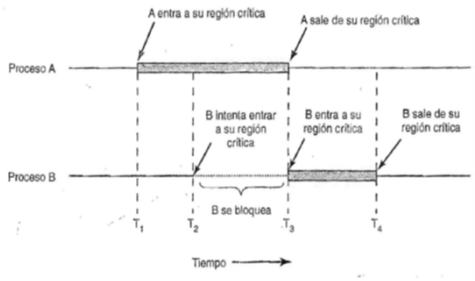
- **REGIONES CRÍTICAS**
- PARA **EVITAR PROBLEMAS** EN EL USO DE **RECURSOS COMPARTIDOS** SE DEBE **PROHIBIR** QUE MÁS DE UN PROCESO LEA Y ESCRIBA LOS DATOS COMPARTIDOS **AL MISMO TIEMPO**:
  - SE NECESITA **EXCLUSIÓN MUTUA**: ASEGURAR QUE **SI UN PROCESO ESTÁ UTILIZANDO** UNA VARIABLE O ARCHIVO COMPARTIDO LOS DEMÁS PROCESOS SE **EXCLUIRÁN** DE HACER LO MISMO.
  - LA **PARTE DEL PROGRAMA** EN LA QUE SE **ACCEDE** A LA **MEMORIA COMPARTIDA** SE CONOCE COMO **REGIÓN CRÍTICA** O SECCIÓN CRÍTICA.
- SI DOS PROCESOS **NUNCA ESTUVIERAN** EN SUS REGIONES CRÍTICAS **AL MISMO TIEMPO** SE **EVITARÍAN** LAS CONDICIONES DE CARRERA.

COMUNICACIÓN ENTRE PROCESOS

- PARA QUE LOS **PROCESOS EN PARALELO COOPEREN** DE LA MANERA **CORRECTA** Y **EFICIENTE** AL UTILIZAR **DATOS COMPARTIDOS** ES NECESARIO CUMPLIR CON CUATRO **CONDICIONES**:
  - **NO PUEDE** HABER DOS PROCESOS **SIMULTÁNEAMENTE** DENTRO DE SUS **REGIONES CRÍTICAS**.
  - **NO PUEDEN** HACERSE **SUPOSICIONES** ACERCA DE LAS VELOCIDADES O EL NÚMERO DE CPUS.
  - **NINGÚN PROCESO** QUE SE EJECUTE **FUERA** DE SU REGIÓN CRÍTICA PUEDE **BLOQUEAR** OTROS PROCESOS.
  - **NINGÚN PROCESO** TIENE QUE **ESPERAR PARA SIEMPRE** PARA ENTRAR A SU **REGIÓN CRÍTICA**.

COMUNICACIÓN ENTRE PROCESOS

- EXCLUSIÓN MUTUA MEDIANTE EL USO DE REGIONES CRÍTICAS.



COMUNICACIÓN ENTRE PROCESOS

- **EXCLUSIÓN MUTUA CON ESPERA OCUPADA**
- EXISTEN VARIAS **PROPOSICIONES** PARA LOGRAR LA **EXCLUSIÓN MUTUA**:
  - MIENTRAS **UN PROCESO ESTÉ ACTUALIZANDO** LA MEMORIA **COMPARTIDA** DESDE SU **REGIÓN CRÍTICA**, **NINGÚN** OTRO PROCESO PUEDE ENTRAR A ***SU PROPIA*** REGIÓN CRÍTICA.
- LAS **PROPOSICIONES** SON:
  - DESHABILITAR INTERRUPCIONES.
  - VARIABLES DE CANDADO.
  - ALTERNANCIA ESTRICTA.
  - SOLUCIÓN DE PETERSON.
  - LA INSTRUCCIÓN TSL.

COMUNICACIÓN ENTRE PROCESOS

- ***DESHABILITAR INTERRUPCIONES***
- EN UN SISTEMA CON **UN SOLO PROCESADOR**, LA SOLUCIÓN MÁS SIMPLE ES HACER QUE CADA PROCESO:
  - **DESHABILITE** TODAS LAS INTERRUPCIONES JUSTO DESPUÉS DE **ENTRAR** A SU REGIÓN CRÍTICA.
  - LAS **REHABILITE** JUSTO DESPUÉS DE **SALIR**.
- CON LAS INTERRUPCIONES DESHABILITADAS **NO PUEDEN** OCURRIR INTERRUPCIONES DE **RELOJ**.
- POR LO GENERAL **NO ES CONVENIENTE** DAR A LOS PROCESOS DE USUARIO EL PODER PARA DESACTIVAR LAS INTERRUPCIONES.
- SI EL SISTEMA ES **MULTIPROCESADOR**, AL DESHABILITAR LAS INTERRUPCIONES **SÓLO** SE AFECTA A LA CPU QUE EJECUTÓ LA DESHABILITACIÓN:
  - LAS DEMÁS CONTINUARÁN EJECUTÁNDOSE Y **PODRÁN ACCEDER** A LA MEMORIA COMPARTIDA.

COMUNICACIÓN ENTRE PROCESOS

- ***VARIABLES DE CANDADO***
- ES UNA SOLUCIÓN DE **SOFTWARE**.
- SE TIENE UNA **VARIABLE COMPARTIDA** (DE CANDADO), QUE AL PRINCIPIO ES 0.
- CUANDO UN PROCESO **DESEA ENTRAR** A SU REGIÓN CRÍTICA PRIMERO **EVALÚA** EL CANDADO:
  - SI EL CANDADO ES **0**, EL PROCESO LO FIJA EN 1 Y **ENTRA** A LA REGIÓN CRÍTICA.
  - SI EL CANDADO YA ES **1 ESPERA** HASTA QUE EL CANDADO SE PONGA EN 0.
- EL PROBLEMA SE PRESENTA CUANDO **MÁS DE UN PROCESO**, APROXIMADAMENTE AL MISMO TIEMPO, VERIFICAN EL CANDADO Y LO ENCUENTRAN EN 0:
  - **VARIOS** PROCESOS SE ENCONTRARÁN EN SUS REGIONES CRÍTICAS AL MISMO TIEMPO.

COMUNICACIÓN ENTRE PROCESOS

- ***ALTERNANCIA ESTRICTA***
- UNA VARIABLE ENTERA, POR EJEMPLO ***TURNO*** (QUE AL PRINCIPIO ES 0), LLEVA LA CUENTA ACERCA DE **A QUÉ PROCESO LE TOCA ENTRAR A SU REGIÓN CRÍTICA** Y EXAMINAR O ACTUALIZAR LA MEMORIA COMPARTIDA.
- EL **PROCESO 0 INSPECCIONA** *TURNO* Y DESCUBRE QUE ES 0 Y ENTRA A SU REGIÓN CRÍTICA PORQUE EL VALOR 0 LO HABILITA. CUANDO SALE DE SU REGIÓN CRÍTICA PONE EL VALOR EN 1.
- EL **PROCESO 1 TAMBIÉN DESCUBRE QUE ES 0** Y POR LO TANTO SE QUEDA EN UN **CICLO EVALUANDO *TURNO*** EN FORMA CONTINUA PARA VER CUÁNDO SE CONVIERTE EN 1 (VALOR QUE HABILITA AL PROCESO 1):
  - A LA EVALUACIÓN EN FORMA CONTINUA DE UNA VARIABLE HASTA QUE APAREZCA CIERTO VALOR SE LE CONOCE COMO ***ESPERA OCUPADA***:
    - **DESPERDICIA** TIEMPO DE LA CPU.

COMUNICACIÓN ENTRE PROCESOS

- UN **CANDADO** QUE UTILIZA LA **ESPERA OCUPADA** ES UN ***CANDADO DE GIRO***.
- LOS PROCESOS **ACCEDEN POR TURNOS** A SUS RESPECTIVAS REGIONES CRÍTICAS:
  - TOMAR TURNOS **NO ES UNA BUENA IDEA** CUANDO UNO DE LOS PROCESOS ES **MUCHO MÁS LENTO** QUE EL OTRO.
- UNA SOLUCIÓN PROPUESTA PARA EL PROBLEMA DE LA REGIÓN CRÍTICA: (a) PROCESO 0. (b) PROCESO 1:

```
while (TRUE) {
    while (turno != 0)      /* ciclo */ ;
    region_critica();
    turno = 1;
    region_nocritica();
}
```

(a)

```
while (TRUE) {
    while (turno != 1)      /* ciclo */ ;
    region_critica();
    turno = 0;
    region_nocritica();
}
```

(b)

COMUNICACIÓN ENTRE PROCESOS

- ***SOLUCIÓN DE PETERSON***
- **ANTES** DE UTILIZAR LAS VARIABLES COMPARTIDAS (ANTES DE ENTRAR A SU REGIÓN CRÍTICA), **CADA PROCESO LLAMA A *ENTRAR_REGION*** CON SU PROPIO NÚMERO DE PROCESO (0 O 1) COMO PARÁMETRO.
- ESTA LLAMADA HARÁ QUE **ESPERE**, SI ES NECESARIO, HASTA QUE SEA **SEGURO** ENTRAR.
- UNA VEZ QUE HAYA TERMINADO CON LAS VARIABLES COMPARTIDAS, EL **PROCESO LLAMA A *SALIR_REGION*** PARA INDICAR QUE HA TERMINADO Y PERMITIR QUE LOS DEMÁS PROCESOS ENTREN, SI ASÍ LO DESEAN.

COMUNICACIÓN ENTRE PROCESOS

- SOLUCIÓN DE PETERSON PARA LOGRAR LA EXCLUSIÓN MUTUA:

```
#define FALSE 0
#define TRUE 1
#define N  2                                    /* número de procesos */

int turno;                                      /* ¿de quién es el turno? */
int interesado[N];                              /* al principio todos los valores son 0 (FALSE) */

void entrar_region(int proceso);                /* el proceso es 0 o 1 */
{
    int otro;                                   /* número del otro proceso */

    otro = 1 - proceso;                         /* el opuesto del proceso */
    interesado[proceso] = TRUE;                 /* muestra que está interesado */
    turno = proceso;                            /* establece la bandera */
    while (turno == proceso && interesado[otro] == TRUE)  /* instrucción nula */;
}

void salir_region(int proceso)                  /* proceso: quién está saliendo */
{
    interesado[proceso] = FALSE;                /* indica que salió de la región crítica */
}
```

COMUNICACIÓN ENTRE PROCESOS

- ***LA INSTRUCCIÓN TSL***
- ESTA PROPOSICIÓN REQUIERE AYUDA DEL **HARDWARE**.
- ALGUNAS COMPUTADORAS TIENEN UNA INSTRUCCIÓN TAL COMO: **TSL REGISTRO, CANDADO**
- **EVALUAR** Y **FIJAR** EL CANDADO FUNCIONA DE LA SIGUIENTE MANERA:
  - **LEE** EL CONTENIDO DE LA PALABRA DE MEMORIA ***CANDADO*** Y LO **GUARDA** EN EL REGISTRO RX.
  - **ALMACENA** UN VALOR DISTINTO DE CERO EN ***CANDADO***.
  - SE GARANTIZA QUE LAS **OPERACIONES** DE LEER LA PALABRA Y ALMACENAR UN VALOR EN ELLA SERÁN **INDIVISIBLES**:
    - **OTRO** PROCESADOR **NO PUEDE ACCEDER** A LA PALABRA DE MEMORIA HASTA QUE TERMINE LA INSTRUCCIÓN.
  - LA CPU QUE **EJECUTA TSL BLOQUEA EL BUS DE MEMORIA** PARA IMPEDIR QUE OTRAS CPUS ACCEDAN A LA MEMORIA HASTA QUE TERMINE.

COMUNICACIÓN ENTRE PROCESOS

- CÓMO ENTRAR Y SALIR DE UNA REGIÓN CRÍTICA MEDIANTE LA INSTRUCCIÓN TSL:

```
entrar_region:
    TSL REGISTRO,CANDADO        |copia candado al registro y fija candado a 1
    CMP REGISTRO,#0             |¿era candado cero?
    JNE entrar_region           |si era distinto de cero, el candado está cerrado, y se repite
    RET                         |regresa al llamador; entra a región crítica

salir_region:
    MOVE CANDADO,#0             |almacena 0 en candado
    RET                         |regresa al llamador
```


COMUNICACIÓN ENTRE PROCESOS

- **DORMIR Y DESPERTAR**
- LAS SOLUCIONES ANTERIORES BASADAS EN ***ESPERA OCUPADA*** **DESPERDICIAN** CICLOS DE PROCESADOR.
- OTRAS **PRIMITIVAS DE COMUNICACIÓN** ENTRE PROCESOS ***BLOQUEAN*** EN VEZ DE DESPERDICIAR CPU AL NO PODER INGRESAR A LAS REGIONES CRÍTICAS.
- UNA DE LAS MÁS SIMPLES ES EL PAR **SLEEP** (**DORMIR**) Y **WAKEUP** (**DESPERTAR**) DE LLAMADAS AL SISTEMA:
  - **SLEEP** HACE QUE EL PROCESO QUE LLAMA SE **BLOQUEE** O **DESACTIVE**, ES DECIR, QUE SE **SUSPENDA** HASTA QUE **OTRO PROCESO LO DESPIERTE**.
  - **WAKEUP** TIENE UN PARÁMETRO, EL PROCESO QUE SE VA A **DESPERTAR** O **ACTIVAR**.
- DE MANERA **ALTERNATIVA**, TANTO SLEEP COMO WAKEUP TIENEN UN PARÁMETRO, UNA **DIRECCIÓN DE MEMORIA** PARA **ASOCIAR** LAS LLAMADAS A **SLEEP** CON LAS LLAMADAS A **WAKEUP**.

COMUNICACIÓN ENTRE PROCESOS

- ***EL PROBLEMA DEL PRODUCTOR-CONSUMIDOR***
- SE CONSIDERA EL PROBLEMA DEL PRODUCTOR-CONSUMIDOR (**PROBLEMA DEL BÚFER LIMITADO**).
- DOS PROCESOS **COMPARTEN** UN BÚFER COMÚN, DE TAMAÑO FIJO.
- EL **PRODUCTOR COLOCA** INFORMACIÓN EN EL BÚFER Y EL **CONSUMIDOR** LA **SACA** (SE PUEDE GENERALIZAR Y QUE HAYA ***M*** PRODUCTORES Y ***N*** CONSUMIDORES).
- EL PROBLEMA SURGE CUANDO EL **PRODUCTOR** DESEA COLOCAR UN **NUEVO ELEMENTO** EN EL BÚFER, PERO ESTE YA SE ENCUENTRA **LLENO**:
  - **EL PRODUCTOR** SE **DESACTIVA** Y LUEGO SE **ACTIVA** CUANDO EL CONSUMIDOR HAYA QUITADO UNO O MÁS ELEMENTOS.
- SI EL **CONSUMIDOR** DESEA QUITAR UN ELEMENTO DEL BÚFER QUE ESTÁ **VACÍO**:
  - SE **DUERME** HASTA QUE EL PRODUCTOR COLOCA ALGO EN EL BÚFER Y LO **DESPIERTA**.

COMUNICACIÓN ENTRE PROCESOS

- EL PROBLEMA DEL PRODUCTOR-CONSUMIDOR CON UNA CONDICIÓN DE CARRERA FATAL:

```
#define N 100                                   /* número de ranuras en el búfer */
int cuenta = 0;                                 /* número de elementos en el búfer */

void productor(void)
{
    int elemento;

    while (TRUE) {                              /* se repite en forma indefinida */
        elemento = producir_elemento();         /* genera el siguiente elemento */
        if (cuenta == N) sleep();               /* si el búfer está lleno, pasa a inactivo */
        insertar_elemento(elemento);            /* coloca elemento en búfer */
        cuenta = cuenta + 1;                    /* incrementa cuenta de elementos en búfer */
        if (cuenta == 1) wakeup(consumidor);    /* ¿estaba vacío el búfer? */
    }
}

void consumidor(void)
{
    int elemento;

    while (TRUE) {                              /* se repite en forma indefinida */
        if (cuenta == 0) sleep();               /* si búfer está vacío, pasa a inactivo */
        elemento = quitar_elemento();           /* saca el elemento del búfer */
        cuenta = cuenta - 1;                    /* disminuye cuenta de elementos en búfer */
        if (cuenta==N-1) wakeup(productor);     /* ¿estaba lleno el búfer? */
        consumir_elemento(elemento);            /* imprime el elemento */
    }
}
```


COMUNICACIÓN ENTRE PROCESOS

- LA **CONDICIÓN DE CARRERA** PUEDE OCURRIR DEBIDO A QUE EL ACCESO A *CUENTA* **NO ESTÁ RESTRINGIDO**.
- ES POSIBLE QUE OCURRA LA SIGUIENTE **SITUACIÓN**:
  - EL BÚFER ESTÁ **VACÍO** Y EL **CONSUMIDOR** ACABA DE LEER *CUENTA* PARA VER SI ES 0.
  - EN ESE INSTANTE, EL PLANIFICADOR DETIENE AL CONSUMIDOR Y EMPIEZA A EJECUTAR EL **PRODUCTOR**.
  - EL PRODUCTOR **INSERTA** UN ELEMENTO EN EL BÚFER, INCREMENTA *CUENTA* Y OBSERVA QUE AHORA ES 1. CONSIDERANDO QUE *CUENTA* ERA ANTES 0, Y QUE EL CONSUMIDOR DEBE ESTAR DORMIDO, LLAMA A *WAKEUP* PARA **DESPERTAR AL CONSUMIDOR**.
  - EL CONSUMIDOR **TODAVÍA NO ESTÁ LÓGICAMENTE DORMIDO** Y LA SEÑAL PARA DESPERTARLO SE **PIERDE**.
  - A SU TURNO EL **CONSUMIDOR** EVALÚA *CUENTA* QUE LEYÓ ANTES, ENCUENTRA QUE ES 0 Y PASA A **DORMIRSE**.
  - CUANDO EL **PRODUCTOR** LLENE EL BÚFER TAMBIÉN SE **DORMIRÁ**.
  - **AMBOS** QUEDARÁN **DORMIDOS** PARA SIEMPRE.

COMUNICACIÓN ENTRE PROCESOS

- **SEMÁFOROS**
- E. W. **DIJKSTRA** (1965) SUGIRIÓ EL USO DE UNA **VARIABLE ENTERA** PARA CONTAR EL NÚMERO DE **SEÑALES DE DESPERTAR** GUARDADAS PARA UN USO FUTURO:
  - INTRODUJO UN NUEVO TIPO DE VARIABLE QUE LLAMÓ **SEMÁFORO**.
  - UN SEMÁFORO PODRÍA TENER:
    - EL **VALOR 0**, INDICANDO QUE **NO SE GUARDARON** SEÑALES DE DESPERTAR .
    - ALGÚN **VALOR POSITIVO** SI ESTUVIERAN **PENDIENTES** UNA O MÁS SEÑALES DE DESPERTAR.
- DIJKSTRA PROPUSO DOS **OPERACIONES:**
  - ***DOWN*** Y ***UP*** (GENERALIZACIONES DE SLEEP Y WAKEUP, RESPECTIVAMENTE).

COMUNICACIÓN ENTRE PROCESOS

- ***DOWN*** COMPRUEBA SI EL VALOR ES **MAYOR QUE 0**:
  - DE SER ASÍ **DISMINUYE** EL VALOR (ES DECIR, UTILIZA UNA SEÑAL DE DESPERTAR ALMACENADA) Y **CONTINUA**.
- SI EL **VALOR ES 0**:
  - EL PROCESO SE **DUERME** SIN COMPLETAR LA OPERACIÓN ***DOWN*** POR EL MOMENTO.
- LAS ACCIONES DE COMPROBAR EL VALOR, MODIFICARLO Y POSIBLEMENTE PASAR A DORMIR, SE REALIZAN EN CONJUNTO COMO UNA SOLA **ACCIÓN ATÓMICA** INDIVISIBLE.
- UNA VEZ QUE **EMPIEZA UNA OPERACIÓN DE SEMÁFORO**, **NINGÚN** OTRO PROCESO PODRÁ **ACCEDER** AL SEMÁFORO SINO HASTA QUE LA OPERACIÓN SE HAYA COMPLETADO O BLOQUEADO.

COMUNICACIÓN ENTRE PROCESOS

- ***UP*** **INCREMENTA** EL VALOR DEL SEMÁFORO DIRECCIONADO.
- SI UNO O MÁS **PROCESOS** ESTABAN **INACTIVOS** EN ESE SEMÁFORO, SIN PODER COMPLETAR UNA OPERACIÓN DOWN ANTERIOR:
  - EL SISTEMA **SELECCIONA** UNO DE ELLOS (AL AZAR) Y PERMITE QUE **COMPLETE SU OPERACIÓN DOWN**.
  - DESPUÉS DE UNA OPERACIÓN ***UP*** EN UN SEMÁFORO QUE CONTENGA PROCESOS DORMIDOS:
    - EL SEMÁFORO SEGUIRÁ EN 0 PERO **HABRÁ UN PROCESO MENOS DORMIDO** EN ÉL.
- LA **OPERACIÓN** DE INCREMENTAR EL SEMÁFORO Y DESPERTAR A UN PROCESO TAMBIÉN ES **INDIVISIBLE**.

COMUNICACIÓN ENTRE PROCESOS

- ***CÓMO RESOLVER EL PROBLEMA DEL PRODUCTOR-CONSUMIDOR MEDIANTE EL USO DE SEMÁFOROS***
- LOS SEMÁFOROS **RESUELVEN** EL PROBLEMA DE **PÉRDIDA DE SEÑALES DE DESPERTAR**.
- PARA QUE FUNCIONEN DE MANERA **CORRECTA** ES ESENCIAL QUE SE IMPLEMENTEN DE UNA FORMA **INDIVISIBLE**.
- LO NORMAL ES IMPLEMENTAR **UP** Y **DOWN** COMO **LLAMADAS AL SISTEMA.**
- SI SE UTILIZAN **VARIAS CPUS**, CADA SEMÁFORO DEBE ESTAR PROTEGIDO POR UNA **VARIABLE DE CANDADO**, EN DONDE SE UTILICEN LAS INSTRUCCIONES **TSL** O **XCHG** PARA ASEGURAR QUE SÓLO **UNA CPU A LA VEZ** PUEDA EXAMINAR EL SEMÁFORO.
- LA OPERACIÓN DE SEMÁFORO SÓLO TARDA UNOS CUANTOS **MICROSEGUNDOS**.

COMUNICACIÓN ENTRE PROCESOS

- SE USAN **TRES SEMÁFOROS**:
  - ***LLENAS*** PARA CONTABILIZAR EL NÚMERO DE RANURAS LLENAS (GARANTIZA **SINCRONIZACIÓN**).
  - ***VACÍAS*** PARA CONTABILIZAR EL NÚMERO DE RANURAS VACÍAS (GARANTIZA **SINCRONIZACIÓN**).
  - ***MUTEX*** PARA ASEGURAR QUE EL PRODUCTOR Y EL CONSUMIDOR NO TENGAN ACCESO AL BÚFER AL MISMO TIEMPO (GARANTIZA LA **EXCLUSIÓN MUTUA**).
- LOS **SEMÁFOROS** QUE SE **INICIALIZAN A 1** Y SON UTILIZADOS POR DOS O MÁS PROCESOS PARA ASEGURAR QUE **SÓLO UNO** DE ELLOS PUEDA ENTRAR A SU **REGIÓN CRÍTICA** EN UN MOMENTO DADO SE LLAMAN **SEMÁFOROS BINARIOS**.

COMUNICACIÓN ENTRE PROCESOS

- EL PROBLEMA DEL PRODUCTOR-CONSUMIDOR MEDIANTE EL USO DE SEMÁFOROS:

```
#define N 100                                   /* número de ranuras en el búfer */
typedef int semaforo;                           /* los semáforos son un tipo especial de int */
semaforo mutex = 1;                             /* controla el acceso a la región crítica */
semaforo vacias = N;                            /* cuenta las ranuras vacías del búfer */
semaforo llenas = 0;                            /* cuenta las ranuras llenas del búfer */

void productor(void)
{
     int elemento;

     while(TRUE){                               /* TRUE es la constante 1 */
          elemento = producir_elemento();       /* genera algo para colocar en el búfer */
          down(&vacias);                        /* disminuye la cuenta de ranuras vacías */
          down(&mutex);                         /* entra a la región crítica */
          insertar_elemento(elemento);          /* coloca el nuevo elemento en el búfer */
          up(&mutex);                           /* sale de la región crítica */
          up(&llenas);                          /* incrementa la cuenta de ranuras llenas */
     }
}

void consumidor(void)
{
     int elemento;

     while(TRUE){                               /* ciclo infinito */
          down(&llenas);                        /* disminuye la cuenta de ranuras llenas */
          down(&mutex);                         /* entra a la región crítica */
          elemento = quitar_elemento();         /* saca el elemento del búfer */
          up(&mutex);                           /* sale de la región crítica */
          up(&vacias);                          /* incrementa la cuenta de ranuras vacías */
          consumir_elemento(elemento);          /* hace algo con el elemento */
     }
}
```

COMUNICACIÓN ENTRE PROCESOS

- **MUTEXES**
- UN **MUTEX** ES UNA VARIABLE QUE PUEDE ESTAR EN UNO DE DOS ESTADOS:
  - ABIERTO (**DESBLOQUEADO**).
  - CERRADO (**BLOQUEADO**).
- SE REQUIERE SÓLO **1 BIT** PARA REPRESENTARLA:
  - PERO EN LA PRÁCTICA SE UTILIZA CON FRECUENCIA UN **ENTERO**, EN DONDE **0** INDICA QUE ESTÁ **ABIERTO** Y TODOS LOS **DEMÁS VALORES** INDICAN QUE ESTÁ **CERRADO**.

COMUNICACIÓN ENTRE PROCESOS

- SE UTILIZAN **DOS PROCEDIMIENTOS** CON LOS MUTEXES:
  - CUANDO UN PROCESO **NECESITA ACCESO** A UNA REGIÓN CRÍTICA LLAMA A ***MUTEX_LOCK***:
    - SI EL MUTEX ESTÁ **ABIERTO** (LO QUE SIGNIFICA QUE LA REGIÓN CRÍTICA ESTÁ **DISPONIBLE**), LA LLAMADA TIENE ÉXITO Y ENTONCES EL PROCESO LLAMADOR PUEDE **ENTRAR** A LA REGIÓN CRÍTICA.
    - SI EL MUTEX YA SE ENCUENTRA **CERRADO**, EL PROCESO QUE HIZO LA LLAMADA SE **BLOQUEA** HASTA QUE EL PROCESO QUE ESTÁ EN LA REGIÓN CRÍTICA TERMINE Y LLAME A ***MUTEX_UNLOCK***.
- SI SE **BLOQUEAN VARIOS PROCESOS** POR EL MUTEX, SE **SELECCIONA** UNO DE ELLOS AL **AZAR** Y SE PERMITE QUE ADQUIERA EL MUTEX.

COMUNICACIÓN ENTRE PROCESOS

- IMPLEMENTACIONES DE *MUTEX_LOCK* Y *MUTEX_UNLOCK:*

```
mutex_lock:
        TSL REGISTRO,MUTEX      |copia el mutex al registro y establece mutex a 1
        CMP REGISTRO,#0         |¿el mutex era 0?
        JZE ok                  |si era cero, el mutex estaba abierto, entonces regresa
        CALL thread_yield       |el mutex está ocupado; planifica otro hilo
        JMP mutex_lock          |intenta de nuevo
ok:     RET                     |regresa al procedimiento llamador; entra a la región crítica

mutex_unlock:
        MOVE MUTEX,#0           |almacena un 0 en el mutex
        RET                     |regresa al procedimiento llamador
```


COMUNICACIÓN ENTRE PROCESOS

- **MONITORES**
- LOS PROGRAMADORES DEBEN SER MUY **CUIDADOSOS** AL UTILIZAR **SEMÁFOROS**:
  - UN LIGERO ERROR Y TODO SE DETIENE EN FORMA ABRUPTA: SE PUEDEN PRODUCIR INTERBLOQUEOS (***DEADLOCK***).
- PARA FACILITAR LA ESCRITURA DE PROGRAMAS CORRECTOS, **BRINCH HANSEN** (1973) Y **HOARE** (1974) PROPUSIERON UNA **PRIMITIVA DE SINCRONIZACIÓN DE MAYOR NIVEL**:
  - EL **MONITOR**.
- UN MONITOR ES UNA **COLECCIÓN DE PROCEDIMIENTOS, VARIABLES Y ESTRUCTURAS DE DATOS** QUE SE AGRUPAN EN UN TIPO ESPECIAL DE MÓDULO O PAQUETE.
- LOS **PROCESOS**:
  - PUEDEN **LLAMAR A LOS PROCEDIMIENTOS** EN UN MONITOR CADA VEZ QUE LO DESEAN.
  - **NO PUEDEN ACCEDER DIRECTAMENTE** A LAS ESTRUCTURAS DE DATOS INTERNAS DEL MONITOR DESDE PROCEDIMIENTOS DECLARADOS FUERA DE ESTE.

COMUNICACIÓN ENTRE PROCESOS

- UN MONITOR:

```
monitor ejemplo
        integer i;
        condition c;

        procedure productor();
        .
        .
        .
        end;

        procedure consumidor();
        .  .  .
        end;
end monitor;
```


COMUNICACIÓN ENTRE PROCESOS

- TIENEN UNA IMPORTANTE **PROPIEDAD** QUE LOS HACE ÚTILES PARA LOGRAR LA **EXCLUSIÓN MUTUA**:
  - SÓLO PUEDE HABER **UN PROCESO ACTIVO** EN UN MONITOR EN CUALQUIER INSTANTE.
- LOS MONITORES SON UNA **CONSTRUCCIÓN DEL LENGUAJE DE PROGRAMACIÓN**:
  - EL **COMPILADOR** SABE QUE SON ESPECIALES Y PUEDE MANEJAR LAS LLAMADAS A LOS PROCEDIMIENTOS DEL MONITOR EN FORMA DISTINTA A LAS LLAMADAS A OTROS PROCEDIMIENTOS.
  - ES **RESPONSABILIDAD DEL COMPILADOR** IMPLEMENTAR LA **EXCLUSIÓN MUTUA** EN LAS ENTRADAS DEL MONITOR, PERO UNA FORMA COMÚN ES UTILIZAR UN MUTEX O SEMÁFORO BINARIO.
- AL CONVERTIR TODAS LAS **REGIONES CRÍTICAS** EN **PROCEDIMIENTOS DE MONITOR**, NUNCA HABRÁ DOS PROCESOS QUE EJECUTEN SUS REGIONES CRÍTICAS AL MISMO TIEMPO.

COMUNICACIÓN ENTRE PROCESOS

- PARA BLOQUEAR LOS PROCESOS QUE NO PUEDEN CONTINUAR SE UTILIZAN **VARIABLES DE CONDICIÓN** Y DOS OPERACIONES: **WAIT** Y **SIGNAL**.
- CUANDO UN PROCEDIMIENTO DE MONITOR DESCUBRE QUE **NO PUEDE CONTINUAR** (POR EJEMPLO, EL PRODUCTOR ENCUENTRA EL BÚFER LLENO), REALIZA UNA OPERACIÓN **WAIT** EN ALGUNA **VARIABLE DE CONDICIÓN**:
  - ESTA ACCIÓN HACE QUE EL PROCESO QUE HACE LA LLAMADA SE **BLOQUEE**.
  - TAMBIÉN **PERMITE QUE OTRO PROCESO** QUE NO HAYA PODIDO ENTRAR AL MONITOR **ENTRE AHORA**.

COMUNICACIÓN ENTRE PROCESOS

- OTRO PROCESO, EL **CONSUMIDOR**, PUEDE **DESPERTAR** A SU SOCIO DORMIDO MEDIANTE LA REALIZACIÓN DE UNA OPERACIÓN **SIGNAL** EN LA **VARIABLE DE CONDICIÓN** QUE SU SOCIO ESTÉ ESPERANDO.
- UNA FORMA DE **EVITAR TENER DOS PROCESOS ACTIVOS** EN EL MONITOR AL MISMO TIEMPO ES REQUERIR QUE UN PROCESO QUE REALICE UNA OPERACIÓN **SIGNAL *DEBA* SALIR** DEL MONITOR DE INMEDIATO:
  - UNA INSTRUCCIÓN **SIGNAL** PODRÍA APARECER SÓLO COMO LA **INSTRUCCIÓN FINAL** EN UN PROCEDIMIENTO DE MONITOR.

COMUNICACIÓN ENTRE PROCESOS

- UN ESQUEMA DEL PROBLEMA **PRODUCTOR-CONSUMIDOR CON MONITORES**:

```
monitor ProductorConsumidor
        condition llenas, vacias;
        integer cuenta;

        procedure insertar(elemento: integer);
        begin
                if cuenta = N then wait(llenas);
                insertar_elemento(elemento);
                cuenta := cuenta + 1;
                if cuenta = 1 then signal(vacias)
        end;

        function eliminar: integer;
        begin
                if cuenta = 0 then wait(vacias);
                eliminar = eliminar_elemento;
                cuenta := cuenta – 1;
                if cuenta = N – 1 then signal(llenas)
        end;

        cuenta := 0;
end monitor;
```


COMUNICACIÓN ENTRE PROCESOS

```
procedure productor;
begin
        while true do
        begin
                elemento = producir_elemento;
                ProductorConsumidor.insertar(elemento)
        End
end;

procedure consumidor;
begin
        while true do
        begin
                elemento = ProductorConsumidor.eliminar;
                consumir_elemento(elemento)
        end
end;
```


COMUNICACIÓN ENTRE PROCESOS

- UNA SOLUCIÓN AL PROBLEMA DEL **PRODUCTOR-CONSUMIDOR EN JAVA**:

```
public class ProductorConsumidor{
  static final int N = 100;          // constante que proporciona el tamaño del búfer
  static productor p = new productor();  // crea instancia de un nuevo hilo productor
  static consumidor c = new consumidor(); // crea instancia de un nuevo hilo consumidor
  static nuestro_monitor mon = new nuestro_monitor();  // crea instancia de un nuevo monitor

  public static void main(String args[]){
    p.start();  // inicia el hilo productor
    c.start();  // inicia el hilo consumidor
  }

  static class productor extends Thread {
    public void run() { // el método run contiene el código del hilo
      int elemento;
      while(true){  // ciclo del productor
        elemento=producir_elemento();
        mon.insertar(elemento);
      }
    }
    private int producir_elemento(){...}  // realmente produce
  }

  static class consumidor extends Thread{
    public void run() { // el método run contiene el código del hilo
      int elemento;
      while(true){  // ciclo del consumidor
        elemento=mon.eliminar();
        consumir_elemento(elemento);
      }
    }
    private void consumir_elemento(int elemento){...}  // realmente consume
  }
```


COMUNICACIÓN ENTRE PROCESOS

```
static class nuestro_monitor{   // este es un monitor
  private int bufer[]=new int[N];
  private int cuenta=0, inf=0, sup=0;  // contadores e índices

  public synchronized void insertar(int val){
    if (cuenta==N) ir_a_estado_inactivo();  // si el búfer está lleno, pasa al estado inactivo
    bufer[sup]=val;  // inserta un elemento en el búfer
    sup=(sup+1)%N;  // ranura en la que se va a colocar el siguiente elemento
    cuenta=cuenta+1;  // ahora hay un elemento más en el búfer
    if (cuenta==1) notify();  // si el consumidor estaba inactivo, lo despierta
  }

  public synchronized int eliminar(){
    int val;
    if (cuenta==0) ir_a_estado_inactivo(); // si el búfer está vacío, pasa al estado inactivo
    val=bufer[inf];  // obtiene un elemento del búfer
    lo = (lo + 1) %N;   // ranura en la que se va a colocar el siguiente elemento
    cuenta=cuenta-1;  // un elemento menos en el búfer
    if (cuenta==N-1) notify();  // si el productor estaba inactivo, lo despierta
    return val;
  }
  private void ir_a_estado_inactivo() {try{wait();}catch(InterruptedException exc){};}
 }
}
```

COMUNICACIÓN ENTRE PROCESOS

- **PASAJE (TRANSMISIÓN) DE MENSAJES**
- EL PASAJE DE MENSAJES (**MESSAGE PASSING**) ES UN MÉTODO DE COMUNICACIÓN ENTRE PROCESOS QUE UTILIZA DOS **PRIMITIVAS** (**SEND** Y **RECEIVE**):
  - AL IGUAL QUE LOS SEMÁFOROS Y A DIFERENCIA DE LOS MONITORES, **SON LLAMADAS AL SISTEMA** EN VEZ DE CONSTRUCCIONES DEL LENGUAJE.

```
send(destino, &mensaje);

y

receive(origen, &mensaje);
```


COMUNICACIÓN ENTRE PROCESOS

- ***ASPECTOS DE DISEÑO PARA LOS SISTEMAS CON PASAJE DE MENSAJES***
- LOS SISTEMAS DE PASO DE MENSAJES TIENEN MUCHOS **PROBLEMAS Y CUESTIONES DE DISEÑO**, EN ESPECIAL SI LOS PROCESOS QUE SE ESTÁN COMUNICANDO SE ENCUENTRAN EN **DISTINTAS MÁQUINAS** CONECTADAS POR UNA RED:
  - SE PUEDEN **PERDER MENSAJES** EN LA RED.
- EL **EMISOR** Y EL **RECEPTOR** PUEDEN **ACORDAR** QUE AL RECIBIR UN MENSAJE, EL RECEPTOR ENVIARÁ DE VUELTA UN MENSAJE DE ACUSE DE RECIBO (**ACKNOWLEDGEMENT**).
- SI EL **EMISOR NO HA RECIBIDO** EL ACUSE DENTRO DE CIERTO **INTERVALO DE TIEMPO**, VUELVE A **TRANSMITIR** EL MENSAJE.
- SE UTILIZAN **NÚMEROS DE SECUENCIA CONSECUTIVOS** EN CADA MENSAJE ORIGINAL.
- LA **AUTENTICACIÓN** TAMBIÉN ES UNA CUESTIÓN EN LOS SISTEMAS DE MENSAJES.

COMUNICACIÓN ENTRE PROCESOS

- ***EL PROBLEMA DEL PRODUCTOR-CONSUMIDOR CON PASAJE DE MENSAJES Y SIN MEMORIA COMPARTIDA***
- SE **SUPONE** QUE:
  - TODOS LOS MENSAJES TIENEN EL **MISMO TAMAÑO**.
  - EL **SISTEMA OPERATIVO** COLOCA LOS MENSAJES ENVIADOS, PERO NO RECIBIDOS, DE MANERA **AUTOMÁTICA**, EN EL BÚFER.
  - SE UTILIZA UN TOTAL DE ***N*** **MENSAJES**.
- EL **CONSUMIDOR** EMPIEZA POR **ENVIAR *N* MENSAJES VACÍOS** AL PRODUCTOR, QUE LOS DEVUELVE LLENOS.
- SE PRESENTAN PROBLEMAS DE **BLOQUEO** CUANDO PRODUCTOR Y CONSUMIDOR TRABAJAN A **DIFERENTES VELOCIDADES**.
- UN SISTEMA DE PASO DE MENSAJES RECONOCIDO ES **MPI** (**MESSAGE-PASSING INTERFACE**: INTERFAZ DE PASAJE DE MENSAJES):
  - SE UTILIZA MUCHO EN LA COMPUTACIÓN CIENTÍFICA.

COMUNICACIÓN ENTRE PROCESOS

- EL PROBLEMA DEL **PRODUCTOR-CONSUMIDOR CON *N* MENSAJES**:

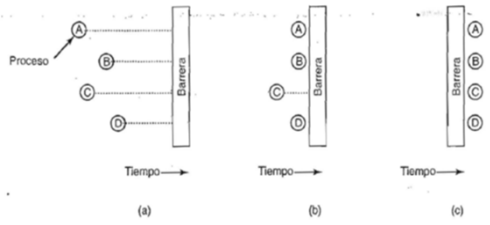
```
#define N 100                                   /* número de ranuras en el búfer */

void productor(void)
{
    int elemento;
    mensaje m;                                  /* búfer de mensajes */

    while (TRUE) {
        elemento=producir_elemento();           /* genera algo para colocar en el búfer */
        receive(consumidor,&m);                 /* espera a que llegue un mensaje vacío */
        crear_mensaje(&m,elemento);             /* construye un mensaje para enviarlo */
        send(consumidor,&m);                    /* envía elemento al consumidor */
    }
}

void consumidor(void)
{
    int elemento, i;
    mensaje m;

    for (i=0; i<N; i++) send(productor,&m);     /* envía N mensajes vacíos */
    while (TRUE){
        receive(productor,&m);                  /* obtiene el mensaje que contiene el elemento */
        elemento=extraer_elemento(&m);          /* extrae el elemento del mensaje */
        send(productor,&m);                     /* envía de vuelta respuesta con mensaje vacío */
        consumir_elemento(elemento);            /* hace algo con el elemento */
    }
}
```


COMUNICACIÓN ENTRE PROCESOS

- **BARRERAS**
- OTRO MECANISMO DE SINCRONIZACIÓN ESTÁ DESTINADO A LOS **GRUPOS DE PROCESOS**, EN VEZ DE LAS SITUACIONES DE TIPO PRODUCTOR-CONSUMIDOR DE DOS PROCESOS.
- ALGUNAS **APLICACIONES SE DIVIDEN EN FASES** Y TIENEN LA REGLA DE QUE **NINGÚN PROCESO PUEDE CONTINUAR** A LA SIGUIENTE FASE SINO HASTA QUE **TODOS LOS PROCESOS ESTÉN LISTOS** PARA HACERLO:
  - SE COLOCA UNA **BARRERA** AL FINAL DE CADA FASE.
- CUANDO UN PROCESO LLEGA A LA BARRERA, SE **BLOQUEA** HASTA QUE TODOS LOS PROCESOS HAN LLEGADO A ELLA.
- CUANDO LOS PROCESOS TERMINAN LA PRIMERA FASE, EJECUTAN LA **PRIMITIVA BARRIER** (GENERALMENTE LLAMANDO A UN PROCEDIMIENTO DE BIBLIOTECA).
- EJ.: LOS PROCESOS PODRÍAN TRABAJAR CON **DISTINTAS PARTES DE UNA MATRIZ GRANDE** (1 MILLÓN X 1 MILLÓN), PUDIENDO PASAR A LA **SIGUIENTE ETAPA** DE PROCESO LUEGO DE HABER PROCESADO LA **TOTALIDAD** DE LA MATRIZ.

COMUNICACIÓN ENTRE PROCESOS

- **USO DE UNA BARRERA**. (A) PROCESOS ACERCÁNDOSE A UNA BARRERA. (B) TODOS LOS PROCESOS, EXCEPTO UNO, BLOQUEADOS EN LA BARRERA. (C) CUANDO EL ÚLTIMO PROCESO LLEGA A LA BARRERA, TODOS SE DEJAN PASAR:



CONCURRENCIA DE EJECUCION Y PLANIFICACION DE PROCESOS

PLANIFICACION DE PROCESOS

- CUANDO **MAS DE UN PROCESO ES EJECUTABLE** DESDE EL PUNTO DE VISTA LOGICO:
  - EL **S. O. DEBE DECIDIR** CUAL DE ELLOS DEBE EJECUTARSE EN PRIMER TERMINO.
  - EL **PLANIFICADOR** ES LA PORCION DEL S. O. QUE DECIDE.
  - EL **ALGORITMO DE PLANIFICACION** ES EL UTILIZADO.

PLANIFICACION DE PROCESOS

- PRINCIPALES **CRITERIOS** RESPECTO DE UN **BUEN ALGORITMO DE PLANIFICACION:**
  - "**EQUIDAD**": GARANTIZAR QUE CADA **PROCESO** OBTIENE SU **PROPORCION JUSTA DE LA CPU**.
  - "**EFICACIA**": MANTENER OCUPADA LA **CPU EL 100%** DEL TIEMPO.
  - "**TIEMPO DE RESPUESTA**": **MINIMIZAR** EL TIEMPO DE RESPUESTA PARA LOS **USUARIOS INTERACTIVOS**.
  - "**TIEMPO DE REGRESO**": **MINIMIZAR** EL TIEMPO QUE DEBEN ESPERAR LOS USUARIOS POR LOTES (**BATCH**) PARA OBTENER SUS RESULTADOS.
  - "**RENDIMIENTO**": **MAXIMIZAR** EL N° DE TAREAS PROCESADAS POR HORA.

PLANIFICACION DE PROCESOS

- ALGUNAS DE ESTAS **METAS SON CONTRADICTORIAS**:
  - EJ.: MINIMIZAR EL TIEMPO DE RESPUESTA PARA LOS USUARIOS INTERACTIVOS SIGNIFICARIA NO EJECUTAR LAS TAREAS BATCH.
- CADA **PROCESO** ES **UNICO E IMPREDECIBLE**:
  - PUEDEN REQUERIR **INTENSIVAMENTE** OPERACIONES DE **E / S** O INTENSIVAMENTE **CPU**.
  - EL PLANIFICADOR DEL **S. O. NO TIENE LA CERTEZA** DE CUANTO TIEMPO TRANSCURRIRA HASTA QUE UN **PROCESO** SE **BLOQUEE**:
    - POR UNA OPERACION DE E / S.
    - POR OTRA RAZON.
- PARA **EVITAR QUE UN PROCESO SE APROPIE DE LA CPU** UN TIEMPO EXCESIVO LOS EQUIPOS POSEEN UN DISPOSITIVO QUE PROVOCA UNA **INTERRUPCION** EN FORMA **PERIODICA**, POR EJ. 60 HZ:
  - 60 VECES POR SEGUNDO.

PLANIFICACION DE PROCESOS

- EN CADA **INTERRUPCION DEL RELOJ EL S. O.** DECIDE:
  - SI EL **PROCESO** QUE SE ESTA EJECUTANDO **CONTINUA**.
  - SI EL **PROCESO** AGOTO SU TIEMPO DE CPU Y DEBE **SUSPENDERSE** Y **CEDER LA CPU** A OTRO PROCESO.
- **PLANIFICACION APROPIATIVA:** ES LA **ESTRATEGIA** DE PERMITIR QUE **PROCESOS EJECUTABLES** (DESDE EL PUNTO DE VISTA LOGICO) SEAN **SUSPENDIDOS** TEMPORALMENTE.
- **PLANIFICACION NO APROPIATIVA:** ES LA **ESTRATEGIA** DE PERMITIR LA EJECUCION DE UN **PROCESO HASTA TERMINAR**.
- **PLANIFICACION DEL PROCESADOR**: DETERMINAR **CUANDO** DEBEN **ASIGNARSE LOS PROCESADORES** Y A **QUE PROCESOS**:
  - ES RESPONSABILIDAD DEL **S. O.**

NIVELES DE PLANIFICACION DEL PROCESADOR

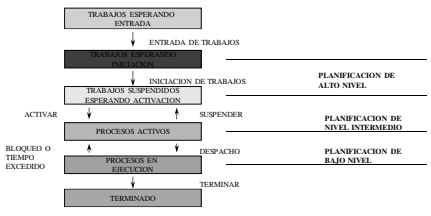
NIVELES DE PLANIFICACION DEL PROCESADOR

- SE CONSIDERAN **TRES NIVELES** IMPORTANTES DE PLANIFICACION.
- "**PLANIFICACION DE ALTO NIVEL**":
  - TAMBIEN SE DENOMINA "**PLANIFICACION DE TRABAJOS**".
  - DETERMINA A **QUE TRABAJOS** SE LES VA A PERMITIR COMPETIR ACTIVAMENTE POR LOS RECURSOS DEL SISTEMA:
    - "**PLANIFICACION DE ADMISION**".
- "**PLANIFICACION DE NIVEL INTERMEDIO**":
  - DETERMINA A **QUE PROCESOS** SE LES PUEDE PERMITIR COMPETIR POR LA CPU.
  - RESPONDE A FLUCTUACIONES A CORTO PLAZO EN LA CARGA DEL SISTEMA:
    - EFECTUA "**SUSPENSIONES**" Y "**ACTIVACIONES**" ("**REANUDACIONES**") DE PROCESOS.
  - DEBE AYUDAR A **ALCANZAR CIERTAS METAS** EN EL RENDIMIENTO TOTAL DEL SISTEMA.

NIVELES DE PLANIFICACION DEL PROCESADOR

- "**PLANIFICACION DE BAJO NIVEL**":
  - DETERMINA A **QUE PROCESO LISTO** SE LE ASIGNA LA CPU CUANDO ESTA QUEDA DISPONIBLE Y ASIGNA LA CPU AL MISMO:
    - "**DESPACHA**" LA CPU AL PROCESO.
  - LA EFECTUA EL "**DESPACHADOR**" DEL S. O.:
    - OPERA **MUCHAS VECES** POR SEGUNDO.
    - RESIDE SIEMPRE EN EL **ALMACENAMIENTO PRIMARIO**.
- LOS DISTINTOS S. O. UTILIZAN VARIAS "**POLITICAS DE PLANIFICACION**":
  - SE INSTRUMENTAN MEDIANTE "**MECANISMOS DE PLANIFICACION**".

NIVELES DE PLANIFICACION DEL PROCESADOR



OBJETIVOS DE LA PLANIFICACION

OBJETIVOS DE LA PLANIFICACION

- **SER JUSTA:**
  - TODOS LOS PROCESOS SON **TRATADOS DE IGUAL MANERA**.
  - NINGUN PROCESO ES **POSTERGADO INDEFINIDAMENTE**.
- **MAXIMIZAR LA CAPACIDAD DE EJECUCION:**
  - **MAXIMIZAR EL N° DE PROCESOS** SERVIDOS POR UNIDAD DE TIEMPO.
- **MAXIMIZAR EL N° DE USUARIOS INTERACTIVOS QUE RECIBAN UNOS TIEMPOS DE RESPUESTA ACEPTABLES:**
  - EN UN MAXIMO DE UNOS SEGUNDOS.
- **SER PREDECIBLE:**
  - UN TRABAJO DADO DEBE EJECUTARSE **APROXIMADAMENTE** EN LA MISMA CANTIDAD DE TIEMPO INDEPENDIENTEMENTE DE LA CARGA DEL SISTEMA.
- **MINIMIZAR LA SOBRECARGA:**
  - NO SUELE CONSIDERARSE UN OBJETIVO MUY IMPORTANTE.

OBJETIVOS DE LA PLANIFICACION

- **EQUILIBRAR EL USO DE RECURSOS:**
  - FAVORECER A LOS PROCESOS QUE UTILIZARAN RECURSOS **INFRAUTILIZADOS**.
- **EQUILIBRAR RESPUESTA Y UTILIZACION:**
  - LA MEJOR MANERA DE GARANTIZAR BUENOS TIEMPOS DE RESPUESTA ES DISPONER DE **RECURSOS SUFICIENTES** CUANDO SE NECESITAN:
    - LA **UTILIZACION TOTAL** DE RECURSOS PODRA SER POBRE.
- **EVITAR LA POSTERGACION INDEFINIDA:**
  - SE UTILIZA LA ESTRATEGIA DEL "**ENVEJECIMIENTO**":
    - MIENTRAS UN PROCESO ESPERA POR UN RECURSO SU PRIORIDAD DEBE AUMENTAR:
      - LA PRIORIDAD LLEGARA A SER TAN ALTA QUE EL PROCESO RECIBIRA EL RECURSO ESPERADO.
- **ASEGURAR LA PRIORIDAD:**
  - LOS MECANISMOS DE PLANIFICACION DEBEN **FAVORECER** A LOS PROCESOS CON **PRIORIDADES MAS ALTAS**.

OBJETIVOS DE LA PLANIFICACION

- **DAR PREFERENCIA A LOS PROCESOS QUE MANTIENEN RECURSOS CLAVES:**
  - UN **PROCESO DE BAJA PRIORIDAD** PODRIA MANTENER UN **RECURSO CLAVE**:
    - PUEDE SER **REQUERIDO** POR UN PROCESO DE MAS ALTA PRIORIDAD.
    - SI EL RECURSO ES **NO APROPIATIVO**:
      - EL MECANISMO DE PLANIFICACION DEBE **OTORGAR AL PROCESO UN TRATAMIENTO MEJOR** DEL QUE LE CORRESPONDERIA NORMALMENTE:
        - ES NECESARIO **LIBERAR RAPIDAMENTE** EL RECURSO CLAVE.
- **DAR MEJOR TRATAMIENTO A LOS PROCESOS QUE MUESTREN UN COMPORTAMIENTO DESEABLE:**
  - EJ.: TASA BAJA DE PAGINACION.

OBJETIVOS DE LA PLANIFICACION

- **DEGRADARSE SUAVEMENTE CON CARGAS PESADAS:**
  - UN MECANISMO DE PLANIFICACION **NO DEBE COLAPSAR** CON EL PESO DE UNA **EXIGENTE CARGA** DEL SISTEMA.
  - SE DEBE **EVITAR UNA CARGA EXCESIVA**:
    - **NO PERMITIENDO** QUE SE CREEN NUEVOS PROCESOS CUANDO LA CARGA YA ES PESADA.
    - DANDO **SERVICIO A LA CARGA MAS PESADA** AL PROPORCIONAR UN NIVEL MODERADAMENTE REDUCIDO DE SERVICIO A TODOS LOS PROCESOS.
- **MUCHAS DE ESTAS METAS SE ENCUENTRAN EN CONFLICTO ENTRE SI:**
  - LA **PLANIFICACION** SE CONVIERTE EN UN **PROBLEMA COMPLEJO.**

CRITERIOS DE PLANIFICACION

CRITERIOS DE PLANIFICACION

- PARA **REALIZAR LOS OBJETIVOS** DE LA PLANIFICACION, UN MECANISMO DE PLANIFICACION DEBE **CONSIDERAR LO SIGUIENTE:**
- **LA LIMITACION DE UN PROCESO A LAS OPERACIONES DE E / S:**
  - CUANDO UN PROCESO CONSIGUE LA CPU:
    - ¿ LA UTILIZA SOLO BREVEMENTE ANTES DE GENERAR UNA PETICION DE E / S ?.
- **LA LIMITACION DE UN PROCESO A LA CPU:**
  - CUANDO UN PROCESO OBTIENE LA CPU:
    - ¿ TIENDE A USARLA HASTA QUE EXPIRA SU TIEMPO ?.
- **SI UN PROCESO ES POR LOTE (BATCH) O INTERACTIVO:**
  - LOS USUARIOS INTERACTIVOS DEBEN RECIBIR INMEDIATO SERVICIO PARA GARANTIZAR BUENOS TIEMPOS DE RESPUESTA.

CRITERIOS DE PLANIFICACION

- **¿ QUE URGENCIA TIENE UNA RESPUESTA RAPIDA ?:**
  - EJ.: UN **PROCESO DE TIEMPO REAL** DE UN SISTEMA DE CONTROL QUE SUPERVISE UNA REFINERIA DE COMBUSTIBLE REQUIERE UNA **RESPUESTA RAPIDA**:
    - MAS RAPIDA QUE LA RESPUESTA REQUERIDA POR UN PROCESO EN LOTES (BATCH) QUE DEBERA ENTREGARSE AL DIA SIGUIENTE.
- **LA PRIORIDAD DE UN PROCESO:**
  - A MAYOR PRIORIDAD MEJOR TRATAMIENTO.

CRITERIOS DE PLANIFICACION

- FRECUENTEMENTE **UN PROCESO GENERA FALLOS** (CARENCIAS) **DE PAGINA:**
  - PROBABLEMENTE LOS PROCESOS QUE GENERAN **POCOS FALLOS DE PAGINA** HAYAN ACUMULADO SUS **CONJUNTOS DE TRABAJO** EN EL ALMACENAMIENTO PRINCIPAL.
  - LOS PROCESOS QUE EXPERIMENTAN **GRAN CANTIDAD DE FALLOS DE PAGINA** AUN **NO HAN ESTABLECIDO** SUS CONJUNTOS DE TRABAJO.
  - UN CRITERIO INDICA **FAVORECER A LOS PROCESOS QUE HAN ESTABLECIDO** SUS CONJUNTOS DE TRABAJO.
  - OTRO CRITERIO INDICA **FAVORECER A LOS PROCESOS CON UNA TASA ALTA DE FALLOS DE PAGINA** YA QUE RAPIDAMENTE GENERARAN UNA PETICION DE E / S.

CRITERIOS DE PLANIFICACION

- FRECUENTEMENTE **UN PROCESO HA SIDO APROPIADO POR OTRO** DE MAS ALTA PRIORIDAD:
  - A MENUDO LOS **PROCESOS APROPIADOS** DEBEN RECIBIR UN **TRATAMIENTO MENOS FAVORABLE**.
  - CADA VEZ QUE EL **S. O. ASUME LA SOBRECARGA** PARA HACER EJECUTAR ESTE PROCESO:
    - EL **CORTO TIEMPO DE EJECUCION ANTES DE LA APROPIACION** NO JUSTIFICA LA SOBRECARGA DE HACER EJECUTAR AL PROCESO EN PRIMER LUGAR.
- **¿ CUANTO TIEMPO DE EJECUCION REAL HA RECIBIDO EL PROCESO ?:**
  - UN CRITERIO CONSIDERA QUE DEBE SER **FAVORECIDO UN PROCESO** QUE HA RECIBIDO **MUY POCO TIEMPO DE CPU**.

CRITERIOS DE PLANIFICACION

- **¿ CUANTO TIEMPO ADICIONAL VA A NECESITAR EL PROCESO PARA TERMINAR ?:**
  - LOS TIEMPOS PROMEDIO DE ESPERA PUEDEN REDUCIRSE **PRIORIZANDO LOS PROCESOS** QUE REQUIEREN DE UN **TIEMPO DE EJECUCION MINIMA** PARA SU TERMINACION.
  - **POCAS VECES ES POSIBLE CONOCER** LA CANTIDAD DE TIEMPO ADICIONAL QUE CADA PROCESO NECESITA PARA TERMINAR.

PLANIFICACION APROPIATIVA VERSUS NO APROPIATIVA

PLANIFICACION APROPIATIVA VERSUS NO APROPIATIVA

- **DISCIPLINA DE PLANIFICACION "APROPIATIVA":**
  - UNA VEZ QUE SE LE HA OTORGADO LA CPU A UN PROCESO, LE PUEDE SER RETIRADA.
- **DISCIPLINA DE PLANIFICACION "NO APROPIATIVA":**
  - UNA VEZ QUE SE LE HA OTORGADO LA CPU A UN PROCESO, NO LE PUEDE SER RETIRADA.

PLANIFICACION APROPIATIVA VERSUS NO APROPIATIVA

- LA **PLANIFICACION APROPIATIVA**:
  - ES UTIL CUANDO LOS **PROCESOS DE ALTA PRIORIDAD** REQUIEREN **ATENCION RAPIDA.**
  - ES IMPORTANTE PARA GARANTIZAR **BUENOS TIEMPOS DE RESPUESTA** EN SISTEMAS INTERACTIVOS DE **TIEMPO COMPARTIDO.**
  - TIENE SU **COSTO EN RECURSOS**:
    - EL **INTERCAMBIO DE CONTEXTO** IMPLICA SOBRECARGA.
    - REQUIERE MANTENER **MUCHOS PROCESOS EN EL ALMACENAMIENTO PRINCIPAL**, EN ESPERA DE LA CPU:
      - IMPLICA SOBRECARGA.

PLANIFICACION APROPIATIVA VERSUS NO APROPIATIVA

- LA **PLANIFICACION NO APROPIATIVA**:
  - SIGNIFICA QUE LOS **TRABAJOS "LARGOS" HACEN ESPERAR** A LOS TRABAJOS "CORTOS".
  - LOGRA **MAS EQUIDAD** EN EL TRATAMIENTO DE LOS PROCESOS.
  - LOGRA HACER **MAS PREDECIBLES LOS TIEMPOS** DE RESPUESTA:
    - LOS **TRABAJOS NUEVOS DE PRIORIDAD ALTA NO PUEDEN DESPLAZAR** A LOS TRABAJOS EN ESPERA.
- EL DISEÑO DE UN **MECANISMO APROPIATIVO** HACE NECESARIO CONSIDERAR LAS **ARBITRARIEDADES** DE CASI CUALQUIER ESQUEMA DE **PRIORIDADES**:
  - MUCHAS VECES LAS PROPIAS PRIORIDADES NO SON ASIGNADAS DE FORMA SIGNIFICATIVA.
- **EL MECANISMO DEBERIA SER SENCILLO PERO EFECTIVO Y SIGNIFICATIVO.**

TEMPORIZADOR DE INTERVALOS O RELOJ DE INTERRUPCION

TEMPORIZADOR DE INTERVALOS O RELOJ DE INTERRUPCION

- EL **PROCESO** AL CUAL ESTA **ASIGNADA LA CPU** SE DICE QUE ESTA **EN EJECUCION**:
  - PUEDE SER UN PROCESO DE S. O. O DE USUARIO.
- EL **S. O. DISPONE DE MECANISMOS PARA QUITARLE LA CPU** A UN PROCESO DE USUARIO PARA **EVITAR QUE MONOPOLICE** EL SISTEMA.
- EL S. O. POSEE UN "**RELOJ DE INTERRUPCION**" O "**TEMPORIZADOR DE INTERVALOS**" PARA GENERAR UNA **INTERRUPCION**:
  - EN ALGUN **TIEMPO FUTURO ESPECIFICO** O
  - **DESPUES DE UN TRANSCURSO** DE TIEMPO EN EL FUTURO.
  - LA **CPU** ES ENTONCES **DESPACHADA** HACIA EL **SIGUIENTE PROCESO.**

TEMPORIZADOR DE INTERVALOS O RELOJ DE INTERRUPCION

- UN **PROCESO RETIENE EL CONTROL** DE LA CPU HASTA QUE:
  - LA **LIBERA** VOLUNTARIAMENTE.
  - EL RELOJ LA **INTERRUMPE**.
  - ALGUNA **OTRA INTERRUPCION** ATRAE LA ATENCION DE LA CPU.
- SI EL **RELOJ INTERRUMPE** UN PROCESO DE USUARIO:
  - LA INTERRUPCION CAUSA LA **EJECUCION DEL S. O.**
  - EL **S. O. DECIDE** CUAL SERA EL PROCESO QUE OBTENDRA LA CPU.

TEMPORIZADOR DE INTERVALOS O RELOJ DE INTERRUPCION

- EL **RELOJ DE INTERRUPCION** AYUDA A **GARANTIZAR TIEMPOS** DE RESPUESTA **RAZONABLES** A USUARIOS INTERACTIVOS:
  - **EVITA** QUE EL SISTEMA SE "**CUELGUE**" A UN SOLO USUARIO EN UN CICLO INFINITO.
  - PERMITE QUE LOS **PROCESOS** RESPONDAN A "**EVENTOS DEPENDIENTES DEL TIEMPO**".
- LOS **PROCESOS** QUE NECESITAN UNA **EJECUCION PERIODICA** DEPENDEN DEL **RELOJ DE INTERRUPCION**.

PRIORIDADES

PRIORIDADES

- LAS PRIORIDADES **PUEDEN SER**:
  - ASIGNADAS **AUTOMATICAMENTE** POR EL SISTEMA.
  - ASIGNADAS DESDE EL **EXTERIOR**.
  - **DINAMICAS**.
  - **ESTATICAS**.
  - ASIGNADAS **RACIONALMENTE**.
  - ASIGNADAS **ARBITRARIAMENTE**:
    - UN MECANISMO DEL SISTEMA NECESITA DISTINGUIR ENTRE PROCESOS SIN IMPORTARLE CUAL ES EL MAS IMPORTANTE.

PRIORIDADES

- **PRIORIDADES ESTATICAS VERSUS DINAMICAS:**
  - LAS "**PRIORIDADES ESTATICAS**":
    - **NO CAMBIAN**.
    - LOS MECANISMOS DE IMPLEMENTACION SON **SENCILLOS**.
    - IMPLICAN UNA **SOBRECARGA** RELATIVAMENTE **BAJA**.
    - **NO RESPONDEN A CAMBIOS** EN EL AMBIENTE (CONTEXTO) QUE HARIAN DESEABLE AJUSTAR ALGUNA PRIORIDAD.
  - LAS "**PRIORIDADES DINAMICAS**":
    - **RESPONDEN AL CAMBIO**.
    - LA **PRIORIDAD INICIAL** ASIGNADA A UN PROCESO PUEDE DURAR POCO TIEMPO:
      - LUEGO SE LA **REAJUSTA** A UN MEJOR VALOR.
    - LOS MECANISMOS DE IMPLEMENTACION SON MAS **COMPLICADOS** QUE PARA PRIORIDADES ESTATICAS.
    - IMPLICAN UNA **SOBRECARGA MAYOR** QUE PARA ESQUEMAS ESTATICOS.

PRIORIDADES

- **PRIORIDADES ADQUIRIDAS:**
  - HACE REFERENCIA AL **TRATAMIENTO ESPECIAL** QUE EN SITUACIONES EXCEPCIONALES REQUIERE UN CIERTO **PROCESO**:
    - PUEDE SIGNIFICAR **RESTAR RECURSOS** A LOS RESTANTES PROCESOS.

TIPOS DE PLANIFICACION

TIPOS DE PLANIFICACION

- **PLANIFICACION A PLAZO FIJO:**
- CIERTOS **TRABAJOS** SE PLANIFICAN PARA SER **TERMINADOS EN UN TIEMPO ESPECIFICO O PLAZO FIJO.**
- ES UNA **PLANIFICACION COMPLEJA** DEBIDO A:
  - EL USUARIO DEBE **SUMINISTRAR ANTICIPADAMENTE** UNA LISTA PRECISA DE **RECURSOS NECESARIOS** PARA EL PROCESO:
    - GENERALMENTE **NO SE DISPONE** DE DICHA INFORMACION.
  - LA EJECUCION DEL TRABAJO DE PLAZO FIJO **NO DEBE PRODUCIR UNA GRAVE DEGRADACION DEL SERVIO** A OTROS USUARIOS.

TIPOS DE PLANIFICACION

- EL **SISTEMA** DEBE **PLANIFICAR CUIDADOSAMENTE** SUS NECESIDADES DE **RECURSOS** HASTA EL PLAZO FIJO:
  - SE PUEDE COMPLICAR CON LAS **DEMANDAS DE RECURSOS** DE NUEVOS PROCESOS QUE INGRESEN AL SISTEMA.
- LA CONCURRENCIA DE **VARIOS PROCESOS DE PLAZO FIJO** (ACTIVOS A LA VEZ) PUEDE REQUERIR **METODOS SOFISTICADOS** DE OPTIMIZACION.
- LA **ADMINISTRACION INTENSIVA** DE RECURSOS PUEDE GENERAR UNA CONSIDERABLE **SOBRECARGA ADICIONAL**.

TIPOS DE PLANIFICACION

- **PLANIFICACION GARANTIZADA:**
- SE ESTABLECEN **COMPROMISOS DE DESEMPEÑO** CON EL PROCESO DEL USUARIO:
  - EJ.: SI EXISTEN **"N" PROCESOS** EN EL SISTEMA EL PROCESO DEL USUARIO RECIBIRA CERCA DEL **"1/N" DE LA POTENCIA DE LA CPU.**
- EL SISTEMA DEBE TENER UN **REGISTRO** DEL:
  - **TIEMPO DE CPU** QUE **CADA PROCESO** HA TENIDO DESDE SU ENTRADA AL SISTEMA.
  - **TIEMPO TRANSCURRIDO** DESDE ESA ENTRADA.
- CON LOS **DATOS** ANTERIORES Y EL **REGISTRO DE PROCESOS** EN CURSO DE EJECUCION EL SISTEMA:
  - CALCULA Y DETERMINA **QUE PROCESOS ESTAN MAS ALEJADOS POR DEFECTO** DE LA RELACION "1/N" PROMETIDA.
  - **PRIORIZA LOS PROCESOS QUE HAN RECIBIDO MENOS CPU DE LA PROMETIDA**.

TIPOS DE PLANIFICACION

- **PLANIFICACION DEL PRIMERO EN ENTRAR PRIMERO EN SALIR (FIFO).**
- ES MUY **SIMPLE**:
  - LOS **PROCESOS SE DESPACHAN** DE ACUERDO CON SU **TIEMPO DE LLEGADA** A LA COLA DE LISTOS.
- UNA VEZ QUE EL **PROCESO OBTIENE LA CPU SE EJECUTA HASTA TERMINAR**:
  - ES UNA DISCIPLINA "**NO APROPIATIVA**".
- PUEDE **OCASIONAR** QUE:
  - **PROCESOS LARGOS HAGAN ESPERAR** A PROCESOS CORTOS.
  - **PROCESOS NO IMPORTANTES HAGAN ESPERAR** A PROCESOS IMPORTANTES.

TIPOS DE PLANIFICACION

- ES **MAS PREDECIBLE** QUE OTROS ESQUEMAS.
- **NO PUEDE GARANTIZAR** BUENOS TIEMPOS DE RESPUESTA INTERACTIVOS.
- SUELE UTILIZARSE **INTEGRADO A OTROS ESQUEMAS**:
  - LOS PROCESOS SE DESPACHAN CON ALGUN **ESQUEMA DE PRIORIDAD.**
  - LOS **PROCESOS CON IGUAL PRIORIDAD** SE DESPACHAN "**FIFO**".

TIPOS DE PLANIFICACION

- **PLANIFICACION DE ASIGNACION EN RUEDA (RR: ROUND ROBIN):**
- LOS **PROCESOS**:
  - SE DESPACHAN EN "**FIFO**".
  - DISPONEN DE UNA **CANTIDAD LIMITADA** DE TIEMPO DE CPU:
    - "DIVISION DE TIEMPO" O "**CUANTO**".
- SI UN PROCESO **NO TERMINA ANTES** DE EXPIRAR SU TIEMPO DE CPU:
  - LA CPU ES **APROPIADA**.
  - LA CPU ES OTORGADA AL **SIGUIENTE PROCESO** EN ESPERA.
  - EL PROCESO APROPIADO ES SITUADO AL **FINAL DE LA LISTA** DE LISTOS.
- ES EFECTIVA EN **AMBIENTES DE TIEMPO COMPARTIDO**.
- LA **SOBRECARGA DE LA APROPIACION** SE MANTIENE BAJA MEDIANTE **MECANISMOS EFICIENTES DE INTERCAMBIO DE CONTEXTO** Y CON **SUFICIENTE MEMORIA PRINCIPAL** PARA LOS PROCESOS.

TIPOS DE PLANIFICACION

- **TAMAÑO DEL CUANTO O QUANTUM:**
- LA DETERMINACION DEL **TAMAÑO DEL CUANTO** ES DECISIVA PARA LA OPERACION EFECTIVA DE UN SISTEMA COMPUTACIONAL.
- LOS **INTERROGANTES** SON:
  - ¿ CUANTO **PEQUEÑO O GRANDE** ?.
  - ¿ CUANTO **FIJO O VARIABLE** ?.
  - ¿ CUANTO **IGUAL PARA TODOS** LOS PROCESOS DE USUARIOS O **DETERMINADO POR SEPARADO** PARA C / U DE ELLOS ?.
- SI EL **CUANTO** SE HACE **MUY GRANDE**:
  - CADA PROCESO RECIBE **TODO EL TIEMPO** NECESARIO PARA LLEGAR A SU TERMINACION:
    - LA ASIGNACION EN RUEDA **("RR") DEGENERA EN "FIFO".**

TIPOS DE PLANIFICACION

- SI EL **CUANTO** SE HACE **MUY PEQUEÑO**:
  - LA **SOBRECARGA DEL INTERCAMBIO DE CONTEXTO** SE CONVIERTE EN UN FACTOR DOMINANTE.
  - EL RENDIMIENTO DEL **SISTEMA SE DEGRADA**:
    - LA **MAYOR PARTE DEL TIEMPO DE CPU** SE INVIERTE EN EL INTERCAMBIO DEL PROCESADOR (**CAMBIO DE CONTEXTO**).
    - LOS **PROCESOS** DE USUARIO DISPONEN DE **MUY POCO TIEMPO** DE CPU.

TIPOS DE PLANIFICACION

- EL **CUANTO** DEBE SER LO **SUFICIENTEMENTE GRANDE** COMO PARA PERMITIR QUE LA **GRAN MAYORIA DE LAS PETICIONES** INTERACTIVAS REQUIERAN DE **MENOS TIEMPO** QUE LA DURACION DEL **CUANTO**:
  - EL **TIEMPO** TRANSCURRIDO **DESDE EL OTORGAMIENTO** DE LA CPU A UN **PROCESO** HASTA QUE **GENERA UNA PETICION DE E / S** DEBE SER **MENOR QUE EL CUANTO** ESTABLECIDO:
    - **OCURRIDA LA PETICION** LA CPU PASA A **OTRO PROCESO**.
    - COMO EL **CUANTO ES MAYOR** QUE EL TIEMPO TRANSCURRIDO HASTA LA PETICION DE E / S:
      - LOS **PROCESOS** TRABAJAN AL **MAXIMO DE VELOCIDAD.**
      - SE **MINIMIZA LA SOBRECARGA** DE APROPIACION.
      - SE **MAXIMIZA LA UTILIZACION** DE LA E / S.
- EL **CUANTO OPTIMO VARIA** DE UN SISTEMA A OTRO Y CON LA CARGA.
- UN **VALOR DE REFERENCIA** ES 100 MSEG (MILISEGUNDOS).

TIPOS DE PLANIFICACION

- **PLANIFICACION DEL TRABAJO MAS CORTO PRIMERO (SJF):**
- ES UNA **DISCIPLINA NO APROPIATIVA** Y POR LO TANTO **NO RECOMENDABLE** EN AMBIENTES DE **TIEMPO COMPARTIDO**.
- EL **PROCESO** EN ESPERA CON EL **MENOR TIEMPO ESTIMADO DE EJECUCION** HASTA SU TERMINACION ES EL **SIGUIENTE** EN EJECUTARSE.
- LOS **TIEMPOS PROMEDIO** DE ESPERA SON **MENORES** QUE CON "**FIFO**".
- LOS **TIEMPOS DE ESPERA** SON **MENOS PREDECIBLES** QUE EN "**FIFO**".
- **FAVORECE A LOS PROCESOS CORTOS** EN DETRIMENTO DE LOS LARGOS.
- TIENDE A **REDUCIR** EL N° DE **PROCESOS EN ESPERA** Y EL N° DE PROCESOS QUE **ESPERAN DETRAS DE PROCESOS LARGOS**.

TIPOS DE PLANIFICACION

- REQUIERE UN **CONOCIMIENTO PRECISO DEL TIEMPO** DE EJECUCION DE UN PROCESO, LO QUE GENERALMENTE **SE DESCONOCE.**
- SE PUEDEN **ESTIMAR LOS TIEMPOS** EN BASE A SERIES DE VALORES ANTERIORES.

TIPOS DE PLANIFICACION

- **PLANIFICACION DEL TIEMPO RESTANTE MAS CORTO (SRT):**
- ES LA **CONTRAPARTE APROPIATIVA** DEL SJF.
- ES UTIL EN SISTEMAS DE **TIEMPO COMPARTIDO**.
- EL PROCESO CON EL **TIEMPO ESTIMADO DE EJECUCION MENOR PARA FINALIZAR** ES EL SIGUIENTE EN SER EJECUTADO.
- UN **PROCESO EN EJECUCION PUEDE SER APROPIADO** POR UN NUEVO PROCESO CON UN **TIEMPO ESTIMADO DE EJECUCION MENOR.**
- TIENE **MAYOR SOBRECARGA** QUE LA PLANIFICACION **SJF**.
- DEBE MANTENER UN **REGISTRO DEL TIEMPO DE SERVICIO** TRANSCURRIDO DEL **PROCESO EN EJECUCION**:
  - AUMENTA LA **SOBRECARGA**.

TIPOS DE PLANIFICACION

- LOS **TRABAJOS LARGOS** TIENEN UN **PROMEDIO** Y UNA **VARIANZA** DE LOS **TIEMPOS DE ESPERA** AUN **MAYOR QUE EN SJF.**
- LA **APROPIACION DE UN PROCESO A PUNTO DE TERMINAR** POR OTRO DE **MENOR DURACION** RECIEN LLEGADO PODRIA SIGNIFICAR:
  - UN **MAYOR TIEMPO DE CAMBIO DE CONTEXTO** (ADMINISTRACION DEL PROCESADOR) QUE EL **TIEMPO DE FINALIZACION DEL PRIMERO**.

TIPOS DE PLANIFICACION

- **AL DISEÑARSE LOS S. O. SE DEBE CONSIDERAR CUIDADOSAMENTE LA SOBRECARGA DE LOS MECANISMOS DE ADMINISTRACION DE RECURSOS COMPARANDOLA CON LOS BENEFICIOS ESPERADOS.**
- **PLANIFICACION EL SIGUIENTE CON RELACION DE RESPUESTA MAXIMA (HRN):**
- **CORRIGE** ALGUNAS DE LAS **DEBILIDADES DEL SJF**:
  - **EXCESO DE PERJUICIO** HACIA LOS PROCESOS (TRABAJOS) **LARGOS**.
  - **EXCESO DE FAVORITISMO** HACIA LOS NUEVOS TRABAJOS **CORTOS**.
- ES UNA **DISCIPLINA NO APROPIATIVA**.

TIPOS DE PLANIFICACION

- LA **PRIORIDAD** DE CADA PROCESO ESTA EN **FUNCION**:
  - **NO SOLO DEL TIEMPO DE SERVICIO** DEL TRABAJO.
  - TAMBIEN INFLUYE LA **CANTIDAD DE TIEMPO** QUE EL TRABAJO HA ESTADO **ESPERANDO SER SERVIDO**.
- CUANDO UN PROCESO HA **OBTENIDO LA CPU** CORRE HASTA **TERMINAR**.
- LAS **PRIORIDADES**, QUE SON **DINAMICAS**, SE CALCULAN SEGUN:
  - **PRIORIDAD = (TE + TS) / TS.**
  - TE: TIEMPO DE ESPERA; TS: TIEMPO DE SERVICIO.

TIPOS DE PLANIFICACION

- **PLANIFICACION POR PRIORIDAD:**
- CONSIDERA **FACTORES EXTERNOS** AL PROCESO.
- LAS **IDEAS CENTRALES** SON:
  - CADA PROCESO TIENE **ASOCIADA UNA PRIORIDAD**.
  - EL PROCESO EJECUTABLE CON **MAXIMA PRIORIDAD** ES EL QUE TIENE EL **PERMISO DE EJECUCION**.
- LOS PROCESOS DE **ALTA PRIORIDAD** PODRIAN EJECUTAR **INDEFINIDAMENTE**:
  - EL **PLANIFICADOR** DEL SISTEMA PUEDE **DISMINUIR LA PRIORIDAD** DEL PROCESO EN EJECUCION EN CADA INTERRUPCION DEL RELOJ.
- LAS **PRIORIDADES** TAMBIEN PUEDEN SER **ASIGNADAS DINAMICAMENTE** POR EL SISTEMA PARA LOGRAR CIERTAS **METAS**:
  - RELACIONADAS CON EL **PROCESADOR** O LA **E / S.**

TIPOS DE PLANIFICACION

- LOS **PROCESOS LIMITADOS POR LA E / S** (REQUERIMIENTOS INTENSIVOS DE E / S) OCUPAN MUCHO DE SU TIEMPO EN **ESPERA DE E / S:**
  - DEBEN TENER **PRIORIDAD PARA USAR LA CPU** Y EFECTUAR LA SIGUIENTE PETICION DE E / S:
    - SE EJECUTARA (LA E / S) EN PARALELO CON OTRO PROCESO QUE UTILICE LA CPU.
  - **SI DEBEN ESPERAR MUCHO** TIEMPO A LA CPU ESTARAN **OCUPANDO MEMORIA** POR UN TIEMPO INNECESARIO.

TIPOS DE PLANIFICACION

- UN **ALGORITMO SENCILLO** CONSISTE EN ESTABLECER QUE LA **PRIORIDAD SEA 1 / F**:
  - "**F**" ES LA **FRACCION DEL ULTIMO CUANTO UTILIZADO** POR EL PROCESO.
  - UN **PROCESO** QUE UTILICE **2 MSEG DE SU CUANTO DE 100 MSEG** TENDRA **PRIORIDAD 50**.
  - UN **PROCESO** QUE SE EJECUTO **50 MSEG** ANTES DEL BLOQUEO TENDRA **PRIORIDAD 2**.
  - UN **PROCESO** QUE UTILIZO **TODO EL CUANTO** TENDRA **PRIORIDAD 1**.

TIPOS DE PLANIFICACION

- FRECUENTEMENTE LOS **PROCESOS SE AGRUPAN** EN "**CLASES DE PRIORIDAD**":
  - SE UTILIZA LA **PLANIFICACION CON PRIORIDADES** ENTRE LAS **CLASES** Y CON **ROUND ROBIN** (RR) **DENTRO DE CADA CLASE.**
  - SI LAS **PRIORIDADES NO SE REAJUSTAN** EN ALGUN MOMENTO:
    - LOS PROCESOS DE LAS **CLASES DE PRIORIDAD MINIMA** PODRIAN **DEMORARSE INDEFINIDAMENTE**.

TIPOS DE PLANIFICACION

- **COLAS DE RETROALIMENTACION DE NIVELES MULTIPLES:**
- PROPORCIONAN UNA ESTRUCTURA PARA **LOGRAR**:
  - **FAVORECER** TRABAJOS **CORTOS**.
  - **FAVORECER** TRABAJOS **LIMITADOS POR LA E / S** PARA OPTIMIZAR EL USO DE LOS DISPOSITIVOS DE E / S.
  - **DETERMINAR LA NATURALEZA** DE UN TRABAJO LO MAS RAPIDO POSIBLE Y **PLANIFICAR EL TRABAJO** (PROCESO) EN CONSECUENCIA.

TIPOS DE PLANIFICACION

- UN **NUEVO PROCESO** ENTRA EN LA **RED DE LINEA DE ESPERA** AL **FINAL DE LA COLA SUPERIOR**.
- SE MUEVE POR ESTA COLA "**FIFO**" HASTA **OBTENER LA CPU**.
- SI EL TRABAJO **TERMINA** O **ABANDONA LA CPU** PARA ESPERAR POR LA TERMINACION DE UNA OPERACION DE **E / S** O LA TERMINACION DE ALGUN **OTRO SUCESO**:
  - EL TRABAJO **ABANDONA** LA RED DE LINEA DE ESPERA.
- SI SU **CUANTO EXPIRA ANTES** DE ABANDONAR LA CPU VOLUNTARIAMENTE:
  - EL PROCESO SE COLOCA EN LA **PARTE TRASERA DE LA COLA DEL SIGUIENTE NIVEL INFERIOR.**

TIPOS DE PLANIFICACION

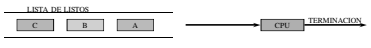
- EL TRABAJO **RECIBE SERVICIO** AL LLEGAR A LA **CABEZA DE ESTA COLA** SI LA PRIMERA ESTA VACIA.
- MIENTRAS EL PROCESO CONTINUE **CONSUMIENDO TOTALMENTE SU CUANTO** EN CADA NIVEL:
  - **CONTINUARA MOVIENDOSE** HACIA EL FINAL DE LAS **COLAS INFERIORES.**
- GENERALMENTE HAY UNA **COLA EN LA PARTE MAS PROFUNDA** A TRAVES DE LA CUAL EL PROCESO CIRCULA EN **ASIGNACION DE RUEDA HASTA QUE TERMINA**.

TIPOS DE PLANIFICACION

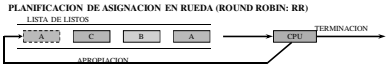
- **EXISTEN ESQUEMAS** EN LOS QUE EL **CUANTO** OTORGADO AL PROCESO **AUMENTA A MEDIDA** QUE EL PROCESO SE MUEVE HACIA LAS **COLAS DE LOS NIVELES INFERIORES**:
  - **CUANTO MAS TIEMPO** HAYA ESTADO EL PROCESO EN LA RED DE LINEA DE ESPERA:
    - **MAYOR SERA SU CUANTO** CADA VEZ QUE OBTIENE LA CPU.
    - **NO PODRA OBTENER** LA CPU MUY A MENUDO DEBIDO A LA **MAYOR PRIORIDAD** DE LOS PROCESOS DE LAS **COLAS SUPERIORES.**
- UN **PROCESO** SITUADO EN UNA **COLA DADA NO PODRA** SER EJECUTADO HASTA QUE LAS **COLAS DE LOS NIVELES SUPERIORES ESTEN VACIAS.**
- UN **PROCESO EN EJECUCION** ES **APROPIADO** POR UN PROCESO QUE LLEGUE A UNA **COLA SUPERIOR**.
- ES UN "**MECANISMO ADAPTABLE**": SE ADAPTA A CARGAS VARIABLES.

TIPOS DE PLANIFICACION

PLANIFICACION PRIMERO EN ENTRAR PRIMERO EN SALIR

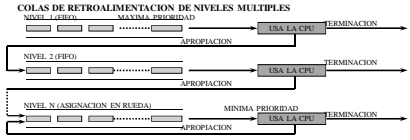


PLANIFICACION DE ASIGNACION EN RUEDA (ROUND ROBIN: RR)



TIPOS DE PLANIFICACION

COLAS DE RETROALIMENTACION DE NIVELES MULTIPLES



TIPOS DE PLANIFICACION

- **POLITICA VERSUS MECANISMO DE PLANIFICACION:**
- PUEDE OCURRIR QUE HAYA **PROCESOS CON MUCHOS PROCESOS HIJOS** EJECUTANDOSE BAJO SU CONTROL:
  - EJ.: **PROCESO EN UN DBMS** CON PROCESOS HIJOS ATENDIENDO FUNCIONES ESPECIFICAS:
    - EJ.: ANALISIS DE INTERROGANTES, ACCESO A DISCOS, ETC.

TIPOS DE PLANIFICACION

- ES POSIBLE QUE EL **PROCESO** PRINCIPAL (**PADRE**) PUEDA IDENTIFICAR LA IMPORTANCIA (O **CRITICIDAD**) DE LOS SUS PROCESOS **HIJOS**:
  - PERO LOS **PLANIFICADORES ANALIZADOS NO ACEPTAN DATOS DE LOS PROCESOS DE USUARIO** RELATIVOS A DECISIONES DE PLANIFICACION.
  - **SOLUCION**: SEPARAR EL **MECANISMO DE PLANIFICACION** DE LA **POLITICA DE PLANIFICACION**:
    - SE **PARAMETRIZA EL ALGORITMO** DE PLANIFICACION.
    - LOS **PARAMETROS** PUEDEN SER DETERMINADOS POR MEDIO DE **PROCESOS DEL USUARIO**.
    - EL **MECANISMO ESTA EN EL NUCLEO** PERO LA **POLITICA** QUEDA ESTABLECIDA POR UN **PROCESO DEL USUARIO**.

TIPOS DE PLANIFICACION

- **PLANIFICACION DE DOS NIVELES:**
- LOS **ESQUEMAS ANALIZADOS** HASTA AHORA SUPONEN QUE **TODOS LOS PROCESOS** EJECUTABLES ESTAN EN LA **MEMORIA PRINCIPAL.**
- SI LA MEMORIA PRINCIPAL ES **INSUFICIENTE**:
  - HABRA **PROCESOS** EJECUTABLES QUE SE MANTENGAN EN **DISCO**.
  - HABRA **IMPORTANTES IMPLICACIONES** PARA LA PLANIFICACION:
    - EL **TIEMPO DE ALTERNANCIA** ENTRE PROCESOS PARA **TRAER** Y PROCESAR UN PROCESO DEL **DISCO** ES **CONSIDERABLEMENTE MAYOR** QUE EL TIEMPO PARA UN PROCESO QUE YA ESTA EN LA **MEMORIA PRINCIPAL**.
    - ES **MAS EFICIENTE EL INTERCAMBIO** DE LOS PROCESOS CON UN **PLANIFICADOR DE DOS NIVELES**.

TIPOS DE PLANIFICACION

- ESQUEMA OPERATIVO DE UN **PLANIFICADOR DE DOS NIVELES**:
  - SE CARGA EN LA **MEMORIA PRINCIPAL** CIERTO **SUBCONJUNTO DE LOS PROCESOS** EJECUTABLES.
  - EL **PLANIFICADOR SE RESTRINGE** A ELLOS DURANTE **CIERTO TIEMPO**.
  - **PERIODICAMENTE** SE LLAMA A UN **PLANIFICADOR DE NIVEL SUPERIOR** PARA:
    - **ELIMINAR DE LA MEMORIA** LOS PROCESOS QUE HAYAN PERMANECIDO EN ELLA EL TIEMPO SUFICIENTE.
    - **CARGAR A MEMORIA** LOS PROCESOS QUE HAYAN ESTADO EN DISCO DEMASIADO TIEMPO.
  - EL **PLANIFICADOR DE NIVEL INFERIOR** SE RESTRINGE DE NUEVO A LOS **PROCESOS EJECUTABLES** QUE SE ENCUENTREN EN LA **MEMORIA**.
  - EL **PLANIFICADOR DE NIVEL SUPERIOR** SE ENCARGA DE **DESPLAZAR LOS PROCESOS** DE MEMORIA A DISCO Y VICEVERSA.

TIPOS DE PLANIFICACION

- **CRITERIOS** QUE PODRIA UTILIZAR EL **PLANIFICADOR DE NIVEL SUPERIOR** PARA TOMAR SUS DECISIONES:
  - ¿ CUANTO **TIEMPO** HA TRANSCURRIDO DESDE EL **ULTIMO INTERCAMBIO DEL PROCESO** ?.
  - ¿ CUANTO **TIEMPO DE CPU** HA UTILIZADO RECIENTEMENTE EL **PROCESO** ?.
  - ¿ **QUE TAN GRANDE** ES EL PROCESO ? (GENERALMENTE LOS PROCESOS PEQUEÑOS NO CAUSAN TANTOS PROBLEMAS EN ESTE SENTIDO).
  - ¿ QUE TAN ALTA ES LA **PRIORIDAD** DEL PROCESO ?.
- EL **PLANIFICADOR DE NIVEL SUPERIOR** PODRIA UTILIZAR CUALQUIERA DE LOS **METODOS DE PLANIFICACION ANALIZADOS**.

MULTIPROCESAMIENTO

MULTIPROCESAMIENTO

- **INTRODUCCION:**
- ES UNA **TENDENCIA SIGNIFICATIVA** EN EL CAMPO DE LA COMPUTACION.
- CONSISTE EN **CONFIGURAR** UN SISTEMA DE COMPUTACION CON **VARIOS PROCESADORES**.
- **NO ES UN ENFOQUE NUEVO** PERO SI POSEE **GRANDES PERSPECTIVAS** EN FUNCION DEL DESARROLLO DE LOS **MICROPROCESADORES**.
- SE PODRIAN CONCEBIR **SISTEMAS CONSTRUIDOS POR CIENTOS O MILES DE MICROPROCESADORES**.

MULTIPROCESAMIENTO

- **CONFIABILIDAD:**
- SI UN **PROCESADOR FALLA** LOS RESTANTES **CONTINUAN** OPERANDO:
  - NO ES AUTOMATICO Y **REQUIERE DE UN DISEÑO CUIDADOSO**.
- UN **PROCESADOR QUE FALLA** HABRA DE **INFORMARLO** A LOS DEMAS DE ALGUNA MANERA PARA QUE SE HAGAN CARGO DE SU TRABAJO.
- LOS **PROCESADORES** EN FUNCIONAMIENTO DEBEN PODER **DETECTAR EL FALLO** DE UN PROCESADOR DETERMINADO.
- EL **S. O. DEBE**:
  - **PERCIBIR** QUE HA FALLADO UN PROCESADOR DETERMINADO:
    - YA NO PODRA ASIGNARLO.
  - **AJUSTAR SUS ESTRATEGIAS** DE ASIGNACION DE RECURSOS PARA EVITAR LA SOBRECARGA DEL SISTEMA QUE ESTA DEGRADADO.

MULTIPROCESAMIENTO

- **EXPLOTACION DEL PARALELISMO:**
- "LA MAYORIA DE LOS **SISTEMAS DE MULTIPROCESAMIENTO** TIENEN COMO **META PRINCIPAL** EL INCREMENTO DE LA **CAPACIDAD DE EJECUCION**".
- LA **PROGRAMACION** SIGUE SIENDO ESENCIALMENTE **SECUENCIAL** Y GENERALMENTE **NO SE EXPLOTA LA CONCURRENCIA.**

MULTIPROCESAMIENTO

- LAS **PRINCIPALES RAZONES** SON:
  - LAS **PERSONAS PIENSAN** EN FORMA **SECUENCIAL**.
  - **NINGUN LENGUAJE HUMANO** PROPORCIONA LA EXPRESION ADECUADA DE **PARALELISMO**:
    - EXISTEN **LENGUAJES DE COMPUTACION** CON SOPORTE DE **CONCURRENCIA** (EJ.: ADA, PASCAL CONCURRENTE, ETC.).
  - **NI EL MULTIPROCESAMIENTO** HA SIDO USADO CON **AMPLITUD** PARA **EXPLOTAR EL PARALELISMO**.
  - EL **HARDWARE TRADICIONAL** DEL COMPUTADOR ESTA ORIENTADO HACIA LA **OPERACION SECUENCIAL**.
  - ES MUY **DIFICIL DEPURAR** PROGRAMAS EN PARALELO.

MULTIPROCESAMIENTO

- LOS **MULTIPROCESADORES** NO SE UTILIZAN A MENUDO PARA EXPLOTAR EL PARALELISMO:
  - ES **MUY ESCASO EL SOFTWARE** QUE EXPLOTE EL **PARALELISMO**.
- LO DESEABLE ES QUE LOS **S. O. Y COMPILADORES** PUEDAN **DETECTAR** E **IMPLEMENTAR** EL **PARALELISMO AUTOMATICAMENTE.**

MULTIPROCESAMIENTO

- **PARALELISMO MASIVO:**
- SE DEBE DISPONER DE **SUFICIENTES PROCESADORES** COMO PARA QUE **TODAS** LAS OPERACIONES QUE PUEDAN SER **EJECUTADAS EN PARALELO** PUEDAN SER ASIGNADAS A **PROCESADORES SEPARADOS**.
- OFRECE UNA FORMA DE **EJECUTAR UN PROGRAMA EN EL MENOR TIEMPO POSIBLE.**
- LA **CUESTION CENTRAL** ES:
  - DISPONIENDO DEL **PARALELISMO MASIVO**:
    - ¿ **CUAL ES EL TIEMPO MINIMO** REQUERIDO PARA EJECUTAR UN ALGORITMO DETERMINADO ?.

MULTIPROCESAMIENTO

- **METAS DE LOS SISTEMAS DE MULTIPROCESAMIENTO:**
- GENERALMENTE SON:
  - **CONFIABILIDAD** Y **DISPONIBILIDAD** MUY ALTAS.
  - **INCREMENTO** DEL PODER DE COMPUTACION.
- SU **DISEÑO MODULAR**:
  - PROPORCIONA UNA **FLEXIBILIDAD** IMPORTANTE.
  - FACILITA LA **EXPANSION** DE LA CAPACIDAD.
- **DETECCION AUTOMATICA DEL PARALELISMO:**
- LOS **MULTIPROCESADORES** HACEN POSIBLE LA **EXPLOTACION DEL PARALELISMO.**
- LOS **SISTEMAS DE COMPUTACION** OBTIENEN LOS BENEFICIOS DEL **PROCESAMIENTO CONCURRENTE**:
  - **MAS** POR LA "**MULTIPROGRAMACION**" DE **VARIOS PROCESOS.**
  - **MENOS** POR LA EXPLOTACION DEL "**PARALELISMO**" DENTRO DE UN **SOLO PROCESO**.

MULTIPROCESAMIENTO

- LA **DETECCION** DEL PARALELISMO:
  - ES UN PROBLEMA **COMPLEJO**.
  - LA **PUEDE EFECTUAR** EL PROGRAMADOR, EL TRADUCTOR DEL LENGUAJE, EL HARDWARE O EL S. O.
- EL PARALELISMO **DENTRO DE LOS PROGRAMAS** PUEDE SER "**EXPLICITO**" O "**IMPLICITO**".

MULTIPROCESAMIENTO

- EL PARALELISMO "**EXPLICITO**":
  - ES **INDICADO DE FORMA ESPECIFICA** POR UN PROGRAMADOR MEDIANTE UNA "**CONSTRUCCION DE CONCURRENCIA**" COMO:
    - COBEGIN;
      - PROPOSICION 1;
      - .....
      - PROPOSICION N;
    - COEND;
  - SE PUEDEN UTILIZAR **PROCESADORES SEPARADOS** PARA EJECUTAR **C / U DE LAS PROPOSICIONES**.
  - ES SUSCEPTIBLE DE **ERRORES DE PROGRAMACION** DIFICILES DE DETECTAR Y DEPURAR.
  - EL **PROGRAMADOR PUEDE OMITIR** TRATAR SITUACIONES DONDE SERIA APLICABLE EL PARALELISMO.

MULTIPROCESAMIENTO

- EL PARALELISMO "**IMPLICITO**":
  - LA VERDADERA ESPERANZA ESTA EN LA **DETECCION AUTOMATICA** DEL PARALELISMO IMPLICITO.
  - ES EL **PARALELISMO INTRINSECO DEL ALGORITMO** PERO **NO ESTABLECIDO EXPLICITAMENTE** POR EL PROGRAMADOR.
  - LOS **COMPILADORES** EXPLOTAN EL PARALELISMO IMPLICITO MEDIANTE LAS **TECNICAS** DE:
    - "**DISTRIBUCION DE CICLOS**".
    - "**REDUCCION DE LA ALTURA DEL ARBOL**".

MULTIPROCESAMIENTO

- **DISTRIBUCION DE CICLOS:**
- UNA "**ESTRUCTURA DE CICLOS O DE REPETICION**" IMPLICA LA **REPETICION DE UNA SERIE DE PROPOSICIONES** (CUERPO DEL CICLO) HASTA QUE OCURRE ALGUNA CONDICION DE TERMINACION.
- EJ.: FOR I = 1 TO 3 DO A (I) = B (I) + C (I);
- EL **PROCESADOR SECUENCIAL** REALIZARA EN SECUENCIA:
  - A (1) = B (1) + C (1); A (2) = B (2) + C (2); A (3) = B (3) + C (3);
- EN UN **SISTEMA DE MULTIPROCESAMIENTO** CON TRES PROCESADORES DISPONIBLES SE PODRIAN EJECUTAR **CONCURRENTEMENTE**.

MULTIPROCESAMIENTO

- UN **COMPILADOR QUE DETECTE AUTOMATICAMENTE** EL PARALELISMO IMPLICITO PUEDE CONVERTIR EL CICLO DEL EJ. EN:
- COBEGIN;
  - A (1) = B (1) + C (1); A (2) = B (2) + C (2); A (3) = B (3) + C (3);
- COEND;
- ESTA TECNICA SE DENOMINA "**DISTRIBUCION DE CICLOS**".

MULTIPROCESAMIENTO

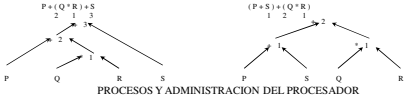
- **REDUCCION DE LA ALTURA DEL ARBOL:**
- UTILIZANDO LAS **PROPIEDADES ASOCIATIVA, CONMUTATIVA Y DISTRIBUTIVA DE LA ARITMETICA**, LOS **COMPILADORES** PUEDEN:
  - **DETECTAR** EL PARALELISMO IMPLICITO EN EXPRESIONES ALGEBRAICAS.
  - **PRODUCIR UN CODIGO OBJETO** PARA MULTIPROCESADORES QUE INDIQUE LAS OPERACIONES QUE SE PUEDEN REALIZAR SIMULTANEAMENTE.
  - **REORDENAR EXPRESIONES** PARA QUE SEAN MAS APROPIADAS PARA LA COMPUTACION EN PARALELO.
- SE INVIERTEN **MAS TIEMPO Y RECURSOS DURANTE LA COMPILACION** PARA **REDUCIR EL TIEMPO DE EJECUCION**:
  - OPTIMIZACION EN EL MOMENTO DE LA COMPILACION PARA LOGRAR EJECUCION EN TIEMPO MINIMO:
    - APLICABLE ESPECIALMENTE CUANDO LOS **SISTEMAS PASAN A PRODUCCION**, NO TANTO CUANDO ESTAN EN DESARROLLO.

MULTIPROCESAMIENTO

IDEA SIMPLIFICADA DE LA ORGANIZACION DE UN MULTIPROCESADOR

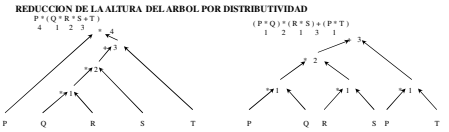
REDUCCION DE LA ALTURA DEL ARBOL POR ASOCIATIVIDAD

REDUCCION DE LA ALTURA DEL ARBOL POR CONMUTATIVIDAD



MULTIPROCESAMIENTO

REDUCCION DE LA ALTURA DEL ARBOL POR DISTRIBUTIVIDAD



REGLA DE "NUNCA ESPERAR":

ES MEJOR DARLE A UN PROCESADOR UNA TAREA QUE PUEDE LLEGAR A NO SER UTILIZADA, QUE TENERLO OCIOSO.

ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

- EL **PROBLEMA CLAVE** ES DETERMINAR LOS **MEDIOS DE CONEXION** DE LOS **PROCESADORES** MULTIPLES Y LOS **PROCESADORES DE E / S** A LAS UNIDADES DE **ALMACENAMIENTO**.
- LOS **MULTIPROCESADORES** SE CARACTERIZAN POR:
  - UN MULTIPROCESADOR CONTIENE **DOS O MAS PROCESADORES** CON CAPACIDADES APROXIMADAMENTE COMPARABLES.
  - TODOS LOS PROCESADORES **COMPARTEN EL ACCESO** A:
    - UN **ALMACENAMIENTO** COMUN.
    - CANALES DE **E / S**, UNIDADES DE CONTROL Y DISPOSITIVOS.
  - TODO ESTA **CONTROLADO POR UN S. O.** QUE PROPORCIONA **INTERACCION ENTRE PROCESADORES Y SUS PROGRAMAS** EN:
    - LOS NIVELES DE TRABAJO, TAREA, PASO, ARCHIVO Y ELEMENTOS DE DATOS.

ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

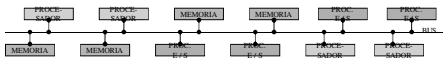
- LAS **ORGANIZACIONES MAS COMUNES** SON:
  - TIEMPO COMPARTIDO O **BUS COMUN** (CONDUCTOR COMUN).
  - MATRIZ DE **BARRAS CRUZADAS** E INTERRUPTORES.
  - ALMACENAMIENTO DE **INTERCONEXION MULTIPLE**.
- **TIEMPO COMPARTIDO O BUS COMUN (O CONDUCTOR COMUN):**
- USA **UN SOLO CAMINO DE COMUNICACION** ENTRE TODAS LAS UNIDADES FUNCIONALES.
- EL **BUS COMUN** ES EN ESENCIA UNA **UNIDAD PASIVA**.
- UN PROCESADOR O PROCESADOR DE E / S QUE DESEE **TRANSFERIR DATOS** DEBE:
  - **VERIFICAR LA DISPONIBILIDAD** DEL CONDUCTOR Y DE LA UNIDAD DE DESTINO.
  - **INFORMAR A LA UNIDAD DE DESTINO** DE LO QUE SE VA A HACER CON LOS DATOS.
  - **INICIAR LA TRANSFERENCIA** DE DATOS.

ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

- LAS **UNIDADES RECEPTORAS** DEBEN PODER:
  - **RECONOCER QUE MENSAJES** DEL BUS SON ENVIADOS HACIA ELLAS.
  - **SEGUIR Y CONFIRMAR LAS SEÑALES DE CONTROL** RECIBIDAS DE LA UNIDAD EMISORA.
- ES UNA **ORGANIZACION ECONOMICA, SIMPLE Y FLEXIBLE** PERO CON **UNA SOLA VIA** DE COMUNICACION:
  - EL SISTEMA **FALLA TOTALMENTE** SI FALLA EL BUS.
  - LA **TASA NETA DE TRANSMISIONES** ESTA **LIMITADA** POR LA TASA NETA DE TRANSMISION DEL **CONDUCTOR**.
  - LA **CONTENCION POR EL USO DEL BUS** EN UN SISTEMA **SOBRECARGADO** PUEDE OCASIONAR UNA SERIA **DEGRADACION**.

ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

ORGANIZACION DE MULTIPROCESADOR DE TIEMPO COMPARTIDO DE BUS COMUN

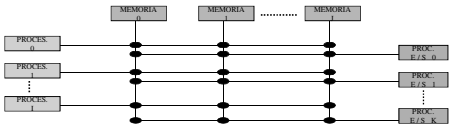


ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

- **MATRIZ DE BARRAS CRUZADAS E INTERRUPTORES:**
- EXISTE UN **CAMINO DIFERENTE** PARA CADA UNIDAD DE **ALMACENAMIENTO**:
  - LAS **REFERENCIAS A DOS UNIDADES DIFERENTES** DE ALMACENAMIENTO **NO SON BLOQUEANTES** SINO SIMULTANEAS.
  - LA **MULTIPLICIDAD** DE CAMINOS DE TRANSMISION PUEDE PROPORCIONAR **TASAS DE TRANSFERENCIA MUY ALTAS**.

ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

ORGANIZACION DEL MULTIPROCESADOR POR MATRIZ DE BARRAS CRUZADAS E INTERRUPTORES



ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

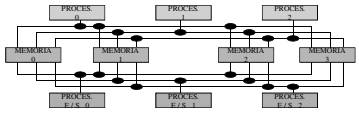
- **ALMACENAMIENTO DE INTERCONEXION MULTIPLE:**
- SE OBTIENE AL **SACAR LAS LOGICAS** DE **CONTROL**, DE **CONMUTACION** Y DE **ARBITRAJE DE PRIORIDADES** FUERA DEL INTERRUPTOR DE BARRAS CRUZADAS:
  - SE LAS **COLOCA EN LA INTERFAZ** DE CADA UNIDAD DE **ALMACENAMIENTO**.
- **CADA UNIDAD** FUNCIONAL PUEDE ACCEDER A **CADA UNIDAD DE ALMACENAMIENTO**, PERO SOLO EN UNA "**CONEXION DE ALMACENAMIENTO**" ESPECIFICA:
  - HAY UNA **CONEXION DE ALMACENAMIENTO POR UNIDAD FUNCIONAL**.

ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

- EL **CONEXIONADO ES MAS COMPLEJO** QUE EN LOS OTROS ESQUEMAS.
- SE PUEDE **RESTRINGIR EL ACCESO A LAS UNIDADES DE ALMACENAMIENTO** PARA QUE NO TODAS LAS UNIDADES DE PROCESAMIENTO LAS ACCEDAN:
  - **HABRA UNIDADES DE ALMACENAMIENTO "PRIVADAS" DE DETERMINADOS PROCESADORES**.

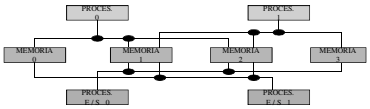
ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

ORGANIZACION DEL MULTIPROCESADOR POR SISTEMA DE MEMORIA DE INTERCONEXION MULTIPLE



ORGANIZACION DEL HARDWARE DEL MULTIPROCESADOR

ORGANIZACION DEL MULTIPROCESADOR POR SISTEMA DE MEMORIA DE INTERCONEXION MULTIPLE CON MEMORIAS PRIVADAS



GRADOS DE ACOPLAMIENTO EN MULTIPROCESAMIENTO

GRADOS DE ACOPLAMIENTO EN MULTIPROCESAMIENTO

- "**MULTIPROCESAMIENTO LIGERAMENTE ACOPLADO**":
  - INCLUYE LA **CONEXION DE DOS O MAS SISTEMAS INDEPENDIENTES** POR MEDIO DE UN **ENLACE DE COMUNICACION.**
  - CADA SISTEMA TIENE SU **PROPIO S. O. Y ALMACENAMIENTO**.
  - LOS SISTEMAS PUEDEN FUNCIONAR **INDEPENDIENTEMENTE**:
    - SE COMUNICAN CUANDO SEA NECESARIO.
  - LOS **SISTEMAS SEPARADOS** PUEDEN:
    - **ACCEDER** A LOS ARCHIVOS DE LOS OTROS.
    - **INTERCAMBIAR TAREAS** A PROCESADORES MENOS CARGADOS.

GRADOS DE ACOPLAMIENTO EN MULTIPROCESAMIENTO

MULTIPROCESAMIENTO LIGERAMENTE ACOPLADO

GRADOS DE ACOPLAMIENTO EN MULTIPROCESAMIENTO

- "**MULTIPROCESAMIENTO RIGIDAMENTE ACOPLADO**":
  - UTILIZA **UN SOLO ALMACENAMIENTO COMPARTIDO** POR VARIOS PROCESADORES.
  - EMPLEA **UN SOLO S. O.** QUE CONTROLA TODOS LOS PROCESADORES Y EL HARDWARE DEL SISTEMA.

GRADOS DE ACOPLAMIENTO EN MULTIPROCESAMIENTO

MULTIPROCESAMIENTO RIGIDAMENTE ACOPLADO

GRADOS DE ACOPLAMIENTO EN MULTIPROCESAMIENTO

- **ORGANIZACION MAESTRO / SATELITE:**
- UN PROCESADOR ESTA DISEÑADO COMO EL "**MAESTRO**" Y LOS OTROS COMO "**SATELITES**".
- EL **PROCESADOR MAESTRO** ES DE **PROPOSITO GRAL.** Y REALIZA:
  - OPERACIONES DE **E / S Y COMPUTACIONES**.
- LOS **PROCESADORES SATELITES** SOLO REALIZAN **COMPUTACIONES**.
- LOS **PROCESOS LIMITADOS POR COMPUTACION** PUEDEN EJECUTARSE CON EFECTIVIDAD EN LOS **SATELITES**.
- LOS **PROCESOS LIMITADOS POR LA E / S** EJECUTADOS EN LOS SATELITES GENERAN FRECUENTES LLAMADAS DE SERVICIOS AL **PROCESADOR MAESTRO**, PUDIENDO RESULTAR INEFICIENTES.

GRADOS DE ACOPLAMIENTO EN MULTIPROCESAMIENTO

- **SI FALLA UN SATELITE** SE PIERDE CAPACIDAD COMPUTACIONAL PERO **EL SISTEMA NO FALLA**.
- **SI FALLA EL MAESTRO EL SISTEMA FALLA** AL NO PODER EFECTUAR OPERACIONES DE E / S:
  - UN **SATELITE DEBERIA ASUMIR** LAS FUNCIONES DEL MAESTRO PREVIO CAMBIO DE LOS PERIFERICOS Y REINICIO DEL SISTEMA.
- **EN EL MULTIPROCESAMIENTO SIMETRICO TODOS PUEDEN HACER E / S.**

S. O. DE MULTIPROCESADORES

S. O. DE MULTIPROCESADORES

- LAS **CAPACIDADES FUNCIONALES** DE LOS **S. O.** DE **MULTIPROGRAMACION** Y DE **MULTIPROCESADORES** INCLUYEN:
  - ASIGNACION Y ADMINISTRACION DE **RECURSOS**.
  - **PROTECCION** DE TABLAS Y CONJUNTOS DE DATOS.
  - **PREVENCION** CONTRA EL INTERBLOQUEO DEL SISTEMA.
  - **TERMINACION** ANORMAL.
  - EQUILIBRIO DE CARGAS DE **ENTRADA / SALIDA**.
  - EQUILIBRIO DE CARGA DEL **PROCESADOR**.
  - **RECONFIGURACION**.

S. O. DE MULTIPROCESADORES

- **LAS TRES ULTIMAS SON ESPECIALMENTE IMPORTANTES EN S. O.** DE **MULTIPROCESADORES**:
  - ES **FUNDAMENTAL**:
    - **EXPLOTAR EL PARALELISMO** EN EL HARDWARE Y EN LOS PROGRAMAS.
    - HACERLO **AUTOMATICAMENTE**.
- LAS **ORGANIZACIONES BASICAS** DE LOS S. O. PARA MULTIPROCESADORES SON:
  - **MAESTRO / SATELITE**.
  - **EJECUTIVO SEPARADO** PARA CADA PROCESADOR.
  - **TRATAMIENTO SIMETRICO** (O ANONIMO) PARA TODOS LOS PROCESADORES.

S. O. DE MULTIPROCESADORES

- **MAESTRO SATELITE:**
- ES LA ORGANIZACION **MAS FACIL DE IMPLEMENTAR**.
- **NO LOGRA** LA UTILIZACION OPTIMA DEL HARDWARE:
  - **SOLO EL PROCESADOR MAESTRO** PUEDE EJECUTAR EL **S. O.**
  - EL **PROCESADOR SATELITE** SOLO PUEDE EJECUTAR **PROGRAMAS DEL USUARIO**.
- LAS **INTERRUPCIONES** GENERADAS POR LOS PROCESOS EN EJECUCION EN LOS **PROCESADORES SATELITES** QUE PRECISAN **ATENCION DEL S. O.:**
  - DEBEN SER **ATENDIDAS POR EL PROCESADOR MAESTRO**.
  - PUEDEN GENERARSE **LARGAS COLAS DE REQUERIMIENTOS** PENDIENTES.

S. O. DE MULTIPROCESADORES

- **EJECUTIVOS SEPARADOS:**
- **CADA PROCESADOR**:
  - TIENE SU **PROPIO S. O.**
  - **RESPONDE A INTERRUPCIONES** DE LOS USUARIOS QUE OPERAN EN ESE PROCESADOR.
- EXISTEN **TABLAS DE CONTROL** CON **INFORMACION GLOBAL** DE TODO EL SISTEMA (EJ.: LISTA DE PROCESADORES CONOCIDOS POR EL S. O.):
  - SE LAS DEBE **ACCEDER UTILIZANDO EXCLUSION MUTUA**.
- ES **MAS CONFIABLE** QUE LA ORGANIZACION MAESTRO / SATELITE.
- **CADA PROCESADOR CONTROLA SUS PROPIOS RECURSOS DEDICADOS.**

S. O. DE MULTIPROCESADORES

- LA **RECONFIGURACION DE LOS DISPOSITIVOS DE E / S** PUEDE:
  - IMPLICAR EL **CAMBIO DE DISPOSITIVOS** A DIFERENTES PROCESADORES CON DISTINTOS S. O.
- LA **CONTENCION SOBRE LAS TABLAS** DEL S. O. ES **MINIMA**.
- LOS **PROCESADORES NO COOPERAN EN LA EJECUCION DE UN PROCESO INDIVIDUAL**, QUE HABRA SIDO ASIGNADO A UNO DE ELLOS.

S. O. DE MULTIPROCESADORES

- **TRATAMIENTO SIMETRICO:**
- ES LA **ORGANIZACION MAS COMPLICADA** DE IMPLEMENTAR Y TAMBIEN **LA MAS PODEROSA Y CONFIABLE**.
- **EL S. O. ADMINISTRA UN GRUPO DE PROCESADORES IDENTICOS**:
  - **CUALQUIERA** PUEDE UTILIZAR **CUALQUIER DISPOSITIVO** DE E / S.
  - **CUALQUIERA** PUEDE REFERENCIAR A **CUALQUIER UNIDAD DE ALMACENAMIENTO.**
- EL **S. O.** PRECISA **CODIGO REENTRANTE** Y **EXCLUSION MUTUA.**
- ES POSIBLE **EQUILIBRAR LA CARGA** DE TRABAJO MAS PRECISAMENTE QUE EN LAS OTRAS ORGANIZACIONES.

S. O. DE MULTIPROCESADORES

- ADQUIEREN SIGNIFICATIVA **IMPORTANCIA** EL HARDWARE Y EL SOFTWARE PARA **RESOLUCION DE CONFLICTOS**.
- **TODOS LOS PROCESADORES PUEDEN COOPERAR** EN LA EJECUCION DE **UN PROCESO DETERMINADO**.
- "**PROCESADOR EJECUTIVO**": ES EL **RESPONSABLE** (UNO SOLO) **EN UN MOMENTO DADO** DE LAS **TABLAS Y FUNCIONES DEL SISTEMA:**
  - SE EVITAN LOS CONFLICTOS SOBRE LA INFORMACION GLOBAL.

RENDIMIENTO DEL SISTEMA DE MULTIPROCESAMIENTO

RENDIMIENTO DEL SISTEMA DE MULTIPROCESAMIENTO

- AUN CON **MULTIPROCESAMIENTO COMPLETAMENTE SIMETRICO** LA **ADICION DE UN NUEVO PROCESADOR NO HARA** QUE LA **CAPACIDAD** DE EJECUCION DEL SISTEMA AUMENTE **SEGUN LA CAPACIDAD** DEL NUEVO PROCESADOR:
  - HAY **SOBRECARGA ADICIONAL** DEL S. O.
  - SE **INCREMENTA LA CONTENCION** POR RECURSOS DEL SISTEMA.
  - HAY **RETRASOS DEL HARDWARE** EN:
    - EL **INTERCAMBIO**.
    - EL **ENCAMINAMIENTO DE LAS TRANSMISIONES** ENTRE UN N° MAYOR DE COMPONENTES.

RENDIMIENTO DEL SISTEMA DE MULTIPROCESAMIENTO

- AL **INCREMENTAR EL N° DE PROCESADORES** "N" SIMILARES EN UN MULTIPROCESADOR, EL **INCREMENTO DE LA PRODUCTIVIDAD:**
  - **NO ES LINEAL.**
  - TIENDE A **DISMINUIR CUANDO "N" CRECE**.

RECUPERACION DE ERRORES

RECUPERACION DE ERRORES

- UNA DE LAS **CAPACIDADES** MAS IMPORTANTES DE LOS S. O. DE **MULTIPROCESADORES** ES LA DE **SOPORTAR FALLAS** DE HARDWARE EN PROCESADORES INDIVIDUALES Y CONTINUAR SU OPERACION.
- **DEBE EXISTIR EL SOPORTE** CORRESPONDIENTE EN EL S. O.
- LAS **TECNICAS DE RECUPERACION** DE ERRORES INCLUYEN:
  - LOS **DATOS CRITICOS** (DEL SISTEMA Y DE USUARIO) DEBEN MANTENERSE EN **COPIAS MULTIPLES** Y EN BANCOS DE ALMACENAMIENTO SEPARADOS.
  - EL **S. O. DEBE EJECUTAR EFECTIVAMENTE** CON LA **CONFIGURACION MAXIMA** Y CON **SUBCONJUNTOS** ANTE FALLAS.
  - DEBE HABER **CAPACIDAD DE DETECCION Y CORRECCION DE ERRORES** DE HARDWARE SIN INTERFERIR CON LA EFICIENCIA OPERACIONAL DEL SISTEMA.
  - SE DEBE **UTILIZAR LA CAPACIDAD OCIOSA DEL PROCESADOR** PARA TRATAR DE **DETECTAR POSIBLES FALLOS** ANTES DE QUE SE PRODUZCAN.
  - EL **S. O.** DEBE DIRIGIR UN **PROCESADOR OPERATIVO** PARA QUE **TOME EL CONTROL** DE UN **PROCESO** QUE SE ESTABA EJECUTANDO EN UN **PROCESADOR QUE FALLA**.

MULTIPROCESAMIENTO SIMETRICO (MPS)

MULTIPROCESAMIENTO SIMETRICO (MPS)

- **CADA PROCESADOR** POSEE **CAPACIDADES FUNCIONALES COMPLETAS**.
- LOS **DISPOSITIVOS DE E / S** PUEDEN SER CONECTADOS A **CADA UNO DE LOS PROCESADORES.**
- TODAS LAS **LLAMADAS AL SUPERVISOR** PUEDEN SER EJECUTADAS EN **TODOS LOS PROCESADORES**:
  - INCLUSIVE LAS DE **E / S**.
- SI UN **PROGRAMA** EN EJECUCION EN UN **PROCESADOR** PIDE UNA **OPERACION DE E / S** EN UN DISPOSITIVO CONECTADO A UN **PROCESADOR DIFERENTE**:
  - EL **PROCESADOR PUEDE CONTINUAR** EJECUTANDO EL TRABAJO.
  - LA **E / S SE COLOCA EN UNA COLA** PARA SU INICIACION POR EL PROCESADOR APROPIADO.

MULTIPROCESAMIENTO SIMETRICO (MPS)

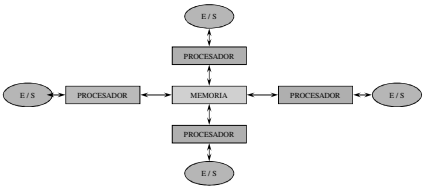
- SE CONSIDERA "**PROCESADOR EJECUTANTE**" AL QUE ESTA EJECUTANDO UN **PROCESO DETERMINADO**.
- SE CONSIDERA "**PROCESADOR PROPIETARIO**" AL QUE ESTA **CONECTADO** A LOS DIFERENTES **DISPOSITIVOS** UTILIZADOS POR EL PROCESO.
- **ES MAS EFICIENTE** QUE LA ORGANIZACION MAESTRO / SATELITE:
  - LOS REQUERIMIENTOS DE **E / S SE ENCOLAN** Y **NO SOBRECARGAN** CON INTERCAMBIO DE CONTEXTO.
  - EN LA **ORGANIZACION MAESTRO / SATELITE** LAS PETICIONES DE **E /S EN EL SATELITE** PROVOCAN UN **INTERCAMBIO DE CONTEXTO EN EL MAESTRO**.

MULTIPROCESAMIENTO SIMETRICO (MPS)

- **CADA PROCESADOR** PUEDE EJECUTAR EL **PLANIFICADOR** PARA BUSCAR EL SIGUIENTE TRABAJO A EJECUTAR:
  - UN **PROCESO DETERMINADO** SE EJECUTA EN **DIFERENTES PROCESADORES** EN DISTINTOS MOMENTOS.
  - MPS **UTILIZA UNA SOLA COLA DE TRABAJOS** Y CADA PROCESADOR PUEDE SELECCIONAR TRABAJOS DE ELLA:
    - SE **EQUILIBRA LA CARGA** ENTRE LOS PROCESADORES.
    - PARA **MINIMIZAR LA CONTENCION** EN EL DESPACHO DE PROCESOS LOS **RELOJES DE LOS PROCESADORES** TIENEN **OBLICUIDAD**:
      - LAS **INTERRUPCIONES** DE RELOJ OCURREN EN **DIFERENTES MOMENTOS**.

MULTIPROCESAMIENTO SIMETRICO (MPS)

EJEMPLO DE IMPLEMENTACION DE MULTIPROCESAMIENTO SIMETRICO



TENDENCIAS DE LOS MULTIPROCESADORES

- TODO INDICA QUE EL USO DE LOS **MULTIPROCESADORES SE INCREMENTARA** CONSIDERABLEMENTE EN EL FUTURO.
- LAS **PRINCIPALES RAZONES** SON:
  - LA **CONFIABILIDAD** REQUERIDA ES CADA VEZ MAYOR.
  - LA **REDUCCION DE COSTOS** CONSECUENCIA DE LOS AVANCES EN MICROELECTRONICA.
  - EL **PREVISIBLE DESARROLLO DE LENGUAJES** QUE PERMITAN A LOS USUARIOS **EXPRESAR EL PARALELISMO EXPLICITAMENTE.**
  - EL PROGRESO EN LA **DETECCION AUTOMATICA DEL PARALELISMO.**

TENDENCIAS DE LOS MULTIPROCESADORES

TENDENCIAS DE LOS MULTIPROCESADORES

- EL HECHO DE QUE SE ESTARIA LLEGANDO A LOS **LIMITES DEL UNIPROCESADOR** DEBIDO A LA **COMPACTACION DE COMPONENTES:**
  - SE ESTARIA PROXIMO A LOS **LIMITES** DE LONGITUD Y DE PROXIMIDAD DE LOS "**CAMINOS ELECTROMAGNETICOS**":
    - LONGITUD DEL RECORRIDO DE LA SEÑAL ELECTROMAGNETICA.
  - ALCANZADOS LOS LIMITES MENCIONADOS LA UNICA POSIBILIDAD DE **INCREMENTAR CAPACIDAD** DE COMPUTO ES MEDIANTE **MULTIPROCESAMIENTO**.

TENDENCIAS DE LOS MULTIPROCESADORES

- EXISTEN ESTUDIOS DE **TENDENCIAS EN ARQUITECTURA** DE COMPUTADORAS QUE APUNTAR A LOS "**POLIPROCESADORES**":
  - SISTEMAS QUE COMBINAN EL **MULTIPROCESAMIENTO**, **SIMETRICO Y ASIMETRICO**, PARA CREAR UNA **JERARQUIA DE PROCESADORES** DENTRO DE UN SISTEMA.