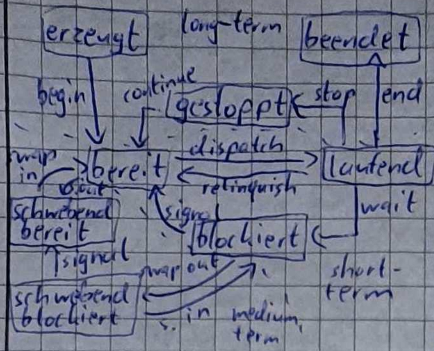
Theorie



Verklemmung
1. ggs. Ausschluss
2. Nachfordern
3. kein Entzug
4. zirk. Abhängigkeit

Monitor
MonitorWS: Ps warten Eintritt
EreignisWS: Ps warten Wartebedingungsende
Hansen: block. alle Signalnehmer bereit
Hoare: block. ein Signalnehmer bereit
Mesa: n. block. einer oder alle

Scheduling
FCFS	RR, VRR	(SPN) HRRN, SRTF	MLQ, MLFQ
koop.	preemptiv	schnellster next	Kombis
Konvoi	Konvoi (RR)	Verhungern	
monopol		Altersgewichtung (HRRN)	
		Bedienzeit + Bereitliste (SRTF)	

Bereitliste
Bereitliste	CPU	Auslastung
global symm.	ident.	gleichmäßig
lokal asym.	versch.	ungleichm.

Synchronisation
einseitig: Auswirkung auf einen Prozess (SiG)
mehrseitig: mehrere Prozesse (Sem)

Pagefault:
0. MMU erkennt Zugriff auf ausgelagerte Seite (Present-bit)
1. MMU löst Trap aus
2. BS löst Page-Fault Handler aus
3. freien Hauptspeicher suchen (ggf. swap)
4. Einlagerung wird angestoßen
5. Zustand -> blockiert
6. Interrupt an CPU signalisiert Einlagerung
7. present-bit -> 1 Seitendeskriptor aktualisieren
8. Zustand -> bereit
9. Zustand -> laufend
10. bei Einlastung Befehl wiederholen

Termine
weich: tolerierbar, benutzbar, mit Zeit wertlos
Fest: tolerierbar, nicht benutzbar
hart: nicht tolerierbar

Inode:
device, inode number, file type & mode, link count, user id, group id, device represented, file size, ..., last access, file creation, last modified, last inode change

generated by github.com/jojodicus/qr2eascii

I'd just like to interject for a moment. What you're referring to as Linux, is in fact, GNU/Linux, or as I've recently taken to calling it, GNU plus Linux. Linux is not an operating system unto itself, but rather another free component of a fully functioning GNU system made useful by the GNU corelibs, ... comprising a full OS as defined by POSIX

yep C

SEM

```
struct SEM {
  volatile val
  pthread_mutex_t m;
  pthread_cond_t c;
}
//create
  calloc
  mutex_init
  cond_init
//P
  lock m
  while(v<=0)
    cond_wait
  v--
  unlock m
//V
  lock m
  v++
  unlock m
```

ENDEVE

```
struct END{
  atomic_int r;
  int w;
  size_t size;
  sem full, free;
  int data[]}
//create
  calloc,
  atomic_init
  sem_create
//put
  P(free)
  data[w]=value;
  w=(w+1)%size;
  V(full),
//get
  P(full)
  do{
    old = atomic_load
    ret=data[old%size];
    new=old+1
    if(new>INT_MAX-B)
      new=new%size
  }while(
    CAS(r,old,new)
  V(free)
  return ret
```

(ein Schreiber)
mehrere -> sems

Signale:

```
//handling
struct sigaction sig = {
  .sa_handler = SIG_IGN/DFL/handler,
  (.sa_flags = SA_RESTART, NOCLDWAIT,)
}; struct sigaction old_sig;
sigemptyset(&sig.sa_mask);
if(sigaction(<SIG>, &sig, &old_sig)) die;
//block
sigset_t mask, old_mask;
sigemptyset(&mask);
if(sigaddset(&mask, <SIG>)) die
if(sigprocmask(SIG_BLOCK, &mask, &old_mask))die;
[while (<cond>)]
  sigsuspend(&old_mask);
if(sigprocmask(SIG_SETMASK, &old_mask, NULL)) die;
//handler
static void handler(int sig){errno=saven;}
```

Networking

```
//server
int sock=socket(AF_INET6, SOCK_STREAM, 0);
if(sock == -1) die
struct sockaddr_in6 dcl = {
  .sin6_family = AF_INET6;
  .sin6_port = htons(<port>),
  .sin6_addr = in6addr_any, };
if(bind(sock, (struct sockaddr*)&dcl, sizeof(dcl)))die
if(listen(sock, SOMAXCON)) die
while(1){
  int fd = accept(sock, 0, 0);
  if(fd == -1){perror, continue;}
  //fdput, handle, close
};close(sock);
//client
struct addrinfo hints = {
  .ai_socktype = SOCK_STREAM,
  .ai_family = AF_UNSPEC,
  .ai_flags = AI_ADDRCONFIG,};
struct addrinfo *res;
int r = getaddrinfo(<host>,<port>,&hints,&res)
if(r){ gai_strerror(r); die }//EAI_SYSTEM -> perror
int sock; struct addrinfo *cur=0;
for(cur=res; cur!=0; cur=cur->ai_next){
  sock = socket(cur->ai_family, cur->ai_socktype,
                cur->ai_protocol);
  if(sock==-1)continue;
  if(connect(sock, cur->ai_addr, cur->ai_addrlen)==0) break;
  close(sock); }
freeaddrinfo(res);
if(!cur) die;
... use socket
//dup2
int dupfd = dup(fd);
if(dupfd==-1) die
//dup2
if(dup2(fd, STDIN_FILENO)==-1)die
... exec
```

Threading

```
//fork
pid_t p = fork();
if(p==-1) die
if(p==0){ //child, exit!}
else{ //parent }
//pthread
pthread_t tid[#];
for(int i=0; i<#; ++i)
  if(errno=pthread_create(&tid[i],0,
                          fkt, arg)) die
//worker
void *fkt(void *arg){
  while(1){...}
}
//join/detach
if(errno=pthread_join(&tid,0)) die
if(errno=pthread_detach(pthread_self())) die
  keine FB am Anfang des threads
```

int event, pid_t p;
if(p=waitpid(-1, &event, WNOHANG)>0) die

1.0

```
//fgets/fputs
char buf[MAXLEN+1];
while(fgets(buf, MAXLEN+1, rx))
  if(fputs(buf, tx)==EOF) break;
//fgetc/fputc
int c;
while((c=fgetc(rx)) != EOF)
  if(fputc(c, tx)==EOF) break;
//ferror, close
if(ferror(rx)||ferror(tx))die
if(fclose(rx)||fclose(tx)) die
```

```
//walkdir
DIR *dir=opendir(<path>);
if(!dir) die
struct dirent *d;
while(1){
  errno=0;
  d=readdir(dir);
  if(!d){if(errno){perror} break;}
  char entry[strlen(path)+1+strlen(d->d_name)];
  sprintf(entry, "%s/%s", path, d->d_name);
  struct stat sb;
  if(!lstat(entry, &sb)){
    perror, continue;}
  //"." "..." skippen
  // S_ISREG(sb.st_mode)
  // open entry (fdopen) }
if(closedir(dir)) die
```

```
//parse positive int or die
errno=0; char *ep;
long x = strtol(str, &ep, 10);
if(errno)die
if(str==ep || *ep!='\0') fpf, exit
if(x<=0) fpf, exit
if(x>INT_MAX) fpf, exit
return (int) x;
```