Elbette! Aşağıda, daha önce pthread fonksiyonlarını nasıl detaylandırdıysak aynı şekilde, verdiğin fonksiyonları gruplar halinde açıklıyorum. Her fonksiyon çıplak tanımı, açıklaması, parametreleri, önemli noktaları ve gerekiyorsa örnekle birlikte anlatılmıştır.

---

1. READLINE KÜTÜPHANESİ

`char *readline(const char *prompt)`

- **Amaç**: Terminalde kullanıcıdan girdi almak.
- **Parametre**: `prompt` kullanıcıya gösterilecek yazıdır.
- **Dönüş**: Kullanıcının girdisini içeren bir string (malloc ile ayrılır).
- **Kullanım**:

```
char *line = readline("shell> ");
if (line)
    free(line);
```

`void add_history(const char *line)`

- **Amaç**: Girilen komutları history'e ekler (ok tuşlarıyla geri çağrılabilir).
- **Not**: readline kullanılmadan çalışmaz.

`void rl_clear_history(void)`

- **Amaç**: readline tarihçesini temizler.

`void rl_on_new_line(void)`

- **Amaç**: readline tamponunu yeni satıra geçmek için ayarlar.

`void rl_replace_line(const char *text, int clear_undo)`

- **Amaç**: Mevcut satırı verilen yazıyla değiştirir.
- **Parametre**: `clear_undo` 1 ise geri alma bilgisi sıfırlanır.

`void rl_redisplay(void)`

- **Amaç**: readline tamponunu ekranda yeniden gösterir.

---

2. DOSYA VE DİZİN FONKSİYONLARI

`int open(const char *pathname, int flags[, mode_t mode])`

- **Amaç**: Dosyayı açar veya oluşturur.
- **Örnek**: `open("file.txt", O_RDONLY);`

`ssize_t read(int fd, void *buf, size_t count)`

- **Amaç**: Dosya tanıtıcısından veri okur.

`int close(int fd)`

- **Amaç**: Dosya tanıtıcısını kapatır.

`int access(const char *pathname, int mode)`

- **Amaç**: Dosyaya erişim olup olmadığını kontrol eder.

`int unlink(const char *pathname)`

- **Amaç**: Bir dosyayı siler.

`int stat(const char *path, struct stat *buf)` / `lstat` / `fstat`

- **Amaç**: Dosyanın bilgilerini getirir (boyut, yetki vs.).

`DIR *opendir(const char *name)`

- **Amaç**: Dizin açar.

`struct dirent *readdir(DIR *dirp)`

- **Amaç**: Dizin içindeki bir sonraki girdiyi okur.

`int closedir(DIR *dirp)`

- **Amaç**: Dizin içeriğini kapatır.

---

3. PROCESS (SÜREÇ) YÖNETİMİ

`pid_t fork(void)`

- **Amaç**: Yeni bir çocuk süreç oluşturur.
- **Dönüş**: 0 ise çocuk, >0 ise ebeveyn.

`int execve(const char *pathname, char *const argv[], char *const envp[])`

- **Amaç**: Geçerli süreci verilen programla değiştirir.

`pid_t wait(int *status)` / `waitpid` / `wait3` / `wait4`

- **Amaç**: Çocuk süreci bekler, durumu döner.

`void exit(int status)`

- **Amaç**: Süreci verilen durum kodu ile sonlandırır.

---

4. SİNYAL YÖNETİMİ

`void (*signal(int sig, void (*func)(int)))(int)`

- **Amaç**: Bir sinyale verilecek işlemi belirtir.

`int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)`

- **Amaç**: signal yerine kullanılır, daha güvenlidir.

`int sigemptyset(sigset_t *set)` / `sigaddset`

- **Amaç**: Sinyal setini boşaltma/ekleme işlemleri.

`int kill(pid_t pid, int sig)`

- **Amaç**: Belirtilen sürece sinyal gönderir.

---

5. TERMINAL VE TTY FONKSİYONLARI

`int isatty(int fd)`

- **Amaç**: Dosya tanıtıcısının terminal olup olmadığını kontrol eder.

`char *ttyname(int fd)`

- **Amaç**: Bir terminalin adını döner.

`int ttyslot(void)`

- **Amaç**: Terminal slot numarasını döner.

`int ioctl(int fd, unsigned long request, ...)`

- **Amaç**: Aygıtlar üzerinde denetim komutları uygular.

`char *getenv(const char *name)`

- **Amaç**: Ortam değişkenini getirir.

`int tcsetattr(int fd, int optional_actions, const struct termios *termios_p)` / `tcgetattr`

- **Amaç**: Terminal ayarlarını al/güncelle.

`int tgetent(char *bp, const char *name)`

- **Amaç**: terminal veritabanından bilgi çeker.

`int tgetflag(char *id)` / `tgetnum` / `tgetstr`

- **Amaç**: terminal kabiliyetlerini sorgular.

`char *tgoto(const char *cap, int col, int row)`

- **Amaç**: Terminal konumlandırma dizesi.

`int tputs(const char *str, int affcnt, int (*putc)(int))`

- **Amaç**: terminal dizelerini yazdırmak için kullanılır.

---

6. DİĞER STANDART FONKSİYONLAR

`int printf(const char *format, ...)`

- Biçimli çıktı basar.

`void *malloc(size_t size)` / `void free(void *ptr)`

- Dinamik bellek ayırma ve serbest bırakma.

`int dup(int oldfd)` / `int dup2(int oldfd, int newfd)`

- Dosya tanıtıcısını kopyalar.

`char *strerror(int errnum)` / `void perror(const char *s)`

- Hata mesajlarını verir.

Her fonksiyonun kullanımıyla ilgili ayrı ayrı örnekler istersen ya da spesifik bir grup üzerine çalışmak istersen ayrıntılandırabilirim.