```
//*************************************************
//*内存乒乓 BUFF 机制
//*算法是，1 Buff ->write,1 Buff ->read,写满后反过来。
//*正常返回 0，出错返回-1
//*************************************************
/*ping pong Buff's ID */
typedef     enum{
    BUFF_PING = 0,     /*Ping ID  的 Buff*/
    BUFF_PONG = 1,     /*Pong ID  的 Buff*/
    BUFF_PIPONUM
}EN_PINGPONG_ID;
/*ping pong use ststus */
typedef     enum{
    BUFF_WRITEABLE = 0,     /*Buff  可写状态*/
    BUFF_READABLE     = 1    /*Buff  可读状态*/
}EN_USE_STATUS;
/*buff use description*/
typedef     struct     _T_BUFF_USE_DES{
    EN_USE_STATUS    eUseStatus;/*可用状态*/
    char*                          pcHeadAddr;/*首地址*/
    unsigned    int          nBuffSize;     /*Buff size*/
    unsigned    int          nOffset;          /*当前可用偏移*/
}T_BUFF_USE_DES;
/*ping pong buff 的使用描述*/
typedef     struct     _T_PINGPONGBUFF_USE_DES{
    EN_PINGPONG_ID     eCurUseID; /*current use ID*/
```

```
    T_BUFF_USE_DES      tPingBuffUse[BUFF_PIPONUM];/*Ping Buff use status*//*Pong Buff use status*/
}T_PINGPONGBUFF_USE_DES;
/*可读消息结构*/
typedef    struct    _T_MSG_REC2FILE{
    //EN_PINGPONG_ID        enBuffID;/*Buff ID*/
    T_BUFF_USE_DES      *ptBuffUse;/*Buff USE*/
    REC_FILE_DESLIST    *ptFileList; /*Rec File List*/
}T_MSG_REC2FILE;
//*************************************************
//*实现一个简单消息队列
//*************************************************
/* Time Weight Task Process Msg type */
typedef    enum{
    TWT_PINGPONGBUFF_REC = 0,    /*ping pong buff  记录文件消息*/
    TWT_MSGTYPE_NUM
}EN_TWT_MSGTYPE;
/*Twt  消息结构*/
typedef    struct    _T_TWTMSG{
    EN_TWT_MSGTYPE      enMsgType;/*msg type*/
    void*                       pvMsg;/*msg entry*/
    void                        (*pfDestroyMsg)(void* pvMsg);/*回收消息体的方法*/
}T_TWTMSG;
#define    TIMEWEIGHT_TASKQUEUE_SIZE          10
/* Helper struct to hold a queue of Msgs */
typedef  struct        _T_TWTMSG_QUEUE{
  T_TWTMSG    *pvMsg[TIMEWEIGHT_TASKQUEUE_SIZE];
```

```
    unsigned long qwrite;
    unsigned long qread;
    unsigned long overflow;
}T_TWTMSG_QUEUE;

/* Helper macros for accessing Msg queues. */
#define TWT_QUEUE_EMPTY(q) \
        (((q)->qwrite == (q)->qread) ?1:0)

#define TWT_QUEUE_FULL(q) \
        ((((((q)->qwrite + 1) % TIMEWEIGHT_TASKQUEUE_SIZE)) == (q)->qread)?1:0)

/**
  * generate a Msg entity
  *  正常返回消息体的指针，异常返回 NULL
  */
T_TWTMSG*       generateMsg(){
        T_TWTMSG*       ptMsg = NULL;
        if(NULL == (ptMsg = malloc(sizeof(T_TWTMSG)))) return       NULL;
        memset(ptMsg, 0, sizeof(T_TWTMSG));
        return       ptMsg;
}
/**
  * destroy a Msg
  *
  */
```

```
void	destroyMsg(T_TWTMSG*	ptMsg){
	if (NULL != ptMsg->pfDestroyMsg)ptMsg->pfDestroyMsg(ptMsg->pvMsg);
	if (NULL != ptMsg)free(ptMsg);
}
/**
 * free a Msg Queue
 *
 */
void	freeTWTMsgQue(T_TWTMSG_QUEUE*	ptMsgQ){
	if (NULL != ptMsgQ)free(ptMsgQ);
}
/**
 * Init a Msg Queue
 *
 */
T_TWTMSG_QUEUE*	initTWTMsgQue(){
	T_TWTMSG_QUEUE*	ptMsgQ = NULL;
	if (NULL == (ptMsgQ = malloc(sizeof(T_TWTMSG_QUEUE))))goto	_ErrRet;
	memset(ptMsgQ, 0, sizeof(T_TWTMSG_QUEUE));
	return	ptMsgQ;

_ErrRet:
	printf("initTWTMsgQue	Fail!\n");
	freeTWTMsgQue(ptMsgQ);
	return NULL;
}
```

```
/**
 * Pop a pvMsg packet from a Msg packet queue
 *
 * @param q is the packet queue from which to pop the pbuf.
 *
 * @return pointer to pvMsg packet if available, NULL otherswise.
 */
T_TWTMSG*    TWTMsgGet(T_TWTMSG_QUEUE    *q)
{
	T_TWTMSG*		ptMsg;
	//*加锁
	if(TWT_QUEUE_EMPTY(q)) {
		/* Return a NULL pointer if the queue is empty. */
		ptMsg = NULL;
	}else {
		/**
		* The queue is not empty so return the next frame from it
		* and adjust the read pointer accordingly.
		*
		*/
		ptMsg = q->pvMsg[q->qread];
		q->qread = ((q->qread + 1) % TIMEWEIGHT_TASKQUEUE_SIZE);
	}
	//*解锁
	return(ptMsg);
```

```
}

/**
  * Push a pvMsg packet onto a pvMsg packet queue
  *
  * @param p is the pvMsg to push onto the packet queue.
  * @param q is the packet queue.
  *
  * @return 0 if successful, -1 if q is full.
  */
int      TWTMsgSend(T_TWTMSG *p, T_TWTMSG_QUEUE *q)
{
        int ret;
        //*加锁
        if(!TWT_QUEUE_FULL(q)){
                /**
                * The queue isn't full so we add the new frame at the current
                * write position and move the write pointer.
                *
                */
                q->pvMsg[q->qwrite] = p;
                q->qwrite = ((q->qwrite + 1) % TIMEWEIGHT_TASKQUEUE_SIZE);
                ret = 0;
        }else{
                /**
                * The stack is full so we are throwing away this value.     Keep track
```

```
            * of the number of times this happens.
            *
            */
            q->overflow++;
            ret = -1;
        }
        //*解锁
        return(ret);
}
//*****************************************************
//*消息分发机制
//*算法是，
//*正常返回 0，出错返回-1
//*****************************************************
extern int       RecToFileMsgProc(T_MSG_REC2FILE* ptMsg);
int     DispatchMsg(T_TWTMSG    *ptMsg){
        if (NULL == ptMsg)goto    _ErrRet;
        /*dispatch msg*/
        switch(ptMsg->enMsgType){
                case TWT_PINGPONGBUFF_REC:
                        RecToFileMsgProc((T_MSG_REC2FILE*)(ptMsg->pvMsg));/*处理消息*/
                        destroyMsg(ptMsg);/*消毁消息*/
                        break;
                default:
                        printf("DispatchMsg Msgtype Error!\n");
                        break;
```

```
	}
	return	0;

_ErrRet:
	printf("DispatchMsg	Fail!\n");
	return -1;
}

/*buff size*/
#define		PINGPONG_BUFF_BSIZE		0x20000//10*1024*1024/*10M*/
/*ping pong buff*/
//char			gacPINGBUFF[PINGPONG_BUFF_BSIZE];/*Ping Buff*/
//char			gacPONGBUFF[PINGPONG_BUFF_BSIZE];/*Pong Buff*/
//*****************************************************
//*释放 ping pong buff
//*必然成功
//*无返回
//*****************************************************
void	DestroyPingPongBuff(T_PINGPONGBUFF_USE_DES*	ptPingPongBuff){
	int		nLoop;
	if (NULL == ptPingPongBuff)return ;
	for (nLoop=0; nLoop<BUFF_PIPONUM; nLoop++){
		if (NULL != ptPingPongBuff->tPingBuffUse[nLoop].pcHeadAddr)free(ptPingPongBuff->tPingBuffUse[nLoop].pcHeadAddr);
	}
	free(ptPingPongBuff);
}
```

```
//*****************************************************
//*初始化 ping pong buff
//*返回 pign pong buff  的描述指针
//*正常返回 0，出错返回-1
//*****************************************************
T_PINGPONGBUFF_USE_DES*    InitPingPongBuff(unsigned int nBuffSize){
      T_PINGPONGBUFF_USE_DES*       ptBuffDes = NULL;
      int                                                  nLoop = 0;

      /*获取  buff  描述*/
      if (NULL == (ptBuffDes=malloc(sizeof(T_PINGPONGBUFF_USE_DES))))goto    _ErrRet;
      memset(ptBuffDes, 0, sizeof(T_PINGPONGBUFF_USE_DES));

      /*分别初始化 ping  和  pong*/
      for (nLoop=0; nLoop<BUFF_PIPONUM; nLoop++){
            if (NULL == (ptBuffDes->tPingBuffUse[nLoop].pcHeadAddr = malloc(nBuffSize)))goto    _ErrRet;
            ptBuffDes->tPingBuffUse[nLoop].nBuffSize = nBuffSize;
            ptBuffDes->tPingBuffUse[nLoop].nOffset       =    0;
            ptBuffDes->tPingBuffUse[nLoop].eUseStatus=BUFF_WRITEABLE;
      }
      ptBuffDes->eCurUseID    =    BUFF_PING;
      return    ptBuffDes;

_ErrRet:
      printf("InitPingPongBuff    Fail!\n");
```

```
	DestroyPingPongBuff(ptBuffDes);
	return NULL;
}

//*************************************************
//*Reset ping pong buff
//*
//*正常返回 0，出错返回-1
//*************************************************
#define	ResetBuffUse(ptBuffUse)	{\
	ptBuffUse->nOffset		= 0;\
	ptBuffUse->eUseStatus = BUFF_WRITEABLE;\
}

/**
 * generate a file rec Msg
 * 正常返回消息体的指针，异常返回 NULL
 */
T_MSG_REC2FILE*	genFRMsg(T_BUFF_USE_DES	*ptBuffUse, REC_FILE_DESLIST	*ptFileList){
	T_MSG_REC2FILE*	ptRFMsg = NULL;
	if (NULL == (ptRFMsg = malloc(sizeof(T_MSG_REC2FILE))))return NULL;
	ptRFMsg->ptBuffUse = ptBuffUse;
	ptRFMsg->ptFileList = ptFileList;
	return	ptRFMsg;
}
/**
```

```
 * destroy a file rec Msg
 *
 */
void     desFRMsg(void*    ptMsg){
      if (NULL != ptMsg)free(ptMsg);
}

//*****************************************************
//*PingPong buff data record
//*算法是，如果 buff 记满，触发一个消息，令写文件线程进入工作状态
//*正常返回 0，出错返回-1
//*****************************************************
int     PingPongBuffRec(T_PINGPONGBUFF_USE_DES*    ptBuffDes, T_TWTMSG_QUEUE *ptMsgQ,
                                                    REC_FILE_DESLIST* pfFileList, const    char* pcData, unsigned long    nDataLen){
      int                       nLoop;
      T_TWTMSG      *ptRecMsg = NULL;

      /*input protect*/
      if ((NULL == ptBuffDes)||(NULL == pcData))goto _ErrRet;

      /*数据过滤，get what i want*/
      if (0 != DataFilter(&pcData, &nDataLen))return    0;

      /*current buff full*/
      if (ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID].nBuffSize < (ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID].nOffset+nDataLen)) {
            /*修改当前 buff 状态*/
```

```
            ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID].eUseStatus = BUFF_READABLE;
            /*发送消息触发记录线程工作*/
            if (NULL == (ptRecMsg = generateMsg()))goto _ErrRet;
            ptRecMsg->enMsgType = TWT_PINGPONGBUFF_REC;
            ptRecMsg->pvMsg    = genFRMsg(&(ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID]), pfFileList);
            ptRecMsg->pfDestroyMsg = desFRMsg;
            if (0 != TWTMsgSend(ptRecMsg, ptMsgQ))goto _ErrRet;
            /*search    for    write useable    buff as new current    buff*/
            for(nLoop=0; nLoop<BUFF_PIPONUM; nLoop++){
                  if(BUFF_WRITEABLE == ptBuffDes->tPingBuffUse[nLoop].eUseStatus)break;
            }
            if (BUFF_PIPONUM <= nLoop)goto _ErrRet;/*if ping and pong all cannot be written*/
            ptBuffDes->eCurUseID = nLoop;
      }
      /*current buff is full?*/
      if (ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID].nBuffSize < (ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID].nOffset+nDataLen))goto _ErrRet;
      /*store data and update the descripton*/
      memcpy(ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID].pcHeadAddr+ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID].nOffset, pcData, nDataLen);
      ptBuffDes->tPingBuffUse[ptBuffDes->eCurUseID].nOffset += nDataLen;
      return    0;

_ErrRet:
      printf("PingPongBuffRec    Fail!\n");
      return -1;
}
```

```
//*****************************************************
//*Ping Pong Buff  写入文件线程消息处理函数
//*算法是，将 buff 数据写入文件并更新所使用的 buff 描述
//*正常返回 0，出错返回-1
//*****************************************************
int       RecToFileMsgProc(T_MSG_REC2FILE* ptMsg){
      if (NULL == ptMsg)goto      _ErrRet;
      if (BUFF_READABLE != ptMsg->ptBuffUse->eUseStatus)goto      _ErrRet;

      /*对对应接口的合法数据进行记录*/
      //if (0 != RecBuff2File(ptMsg->ptFileList, ptMsg->ptBuffUse->pcHeadAddr, ptMsg->ptBuffUse->nOffset))goto      _ErrRet;
      if (0 != ExRecOutFileList(ptMsg->ptFileList, ptMsg->ptBuffUse->pcHeadAddr, ptMsg->ptBuffUse->nOffset))goto      _ErrRet;

      /*更新 Buff use*/
      ResetBuffUse(ptMsg->ptBuffUse);

      return    0;
_ErrRet:
      printf("RecToFileMsgProc    Fail!\n");
      return    -1;
}


//*****************************************************
//*线程消息处理函数
```

```
//*算法是，获取消息，分发，处理
//*
//*************************************************
#define    TWT_TWC_ms          10
void       TWT_Task(void*    pvParam){
      T_TWTMSG_QUEUE                  *ptMsgQ = pvParam;
      T_TWTMSG                              *ptMsg = NULL;

      while(1){
            /*if msg come?*/
            if (!(ptMsg = TWTMsgGet(ptMsgQ))){
                  //Sleep(TWT_TWC_ms);
                  continue;
            }
            /*dispatch msg*/
            DispatchMsg(ptMsg);
      }
}
```